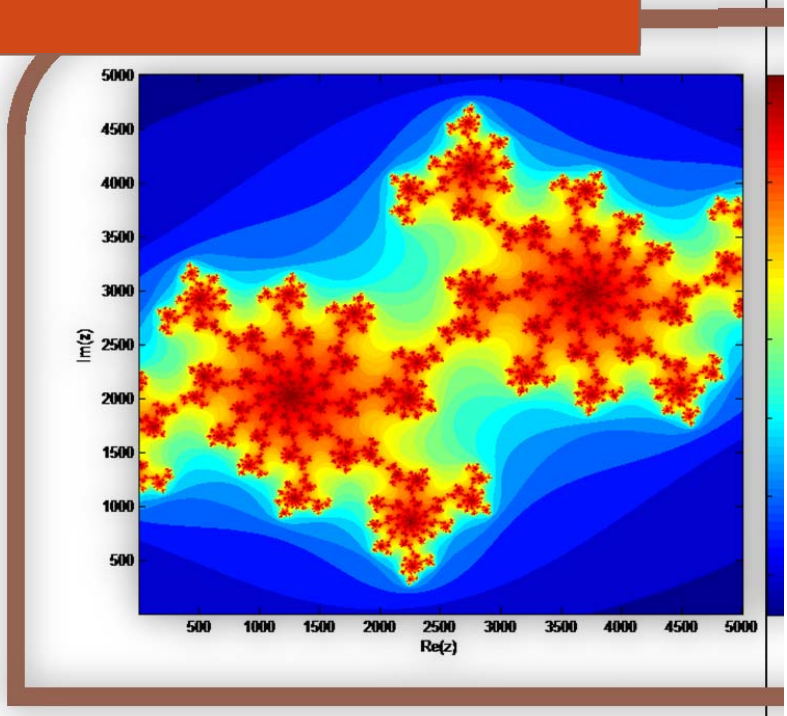
KISHORE VAIGYANIK PROTSAHAN YOJANA
किशोर वैज्ञानिक प्रोत्साहन योजना
NATIONAL FELLOWSHIPS FOR STUDENTS INTERESTED IN RESEARCH CAREERS
KVPY

Mathematical modeling

With applications in problems of Physics and Mathematics



GURSIMRAN SINGH

BACHELOR OF COMPUTER SCIENCE AND ENGINEERING
THAPAR UNIVERSITY, PATIALA

AT
PHYSICS DEPARTMENT
INDIAN INSTITUTE OF TECHNOLOGY BOMBAY

Certificate

This is to certify that the project work titled "**Mathematical modeling with applications in problems of Physics and Mathematics**" has been successfully carried out by Gursimran singh, student of B.E Computer Science and Engineering, Thapar univerity, Patiala under the "KVPY Summer Camp 2011" from June 2 to July 13 2011, under the guidance of Prof. K G Suresh, Physics Department, IIT Bombay.

Prof. K G Suresh
Physics Department
IIT Bombay

Index

Acknowledgements

First of all, I would like to express my sincere thanks to KVPY cell for inviting me for this summer camp. I believe, a student gets an excellent research field after attending this camp.

My whole hearted thanks to Mr. Amit Shinde for his support and help throughout the duration of my stay.

I would like to thank with most emphasis to Prof. K G Suresh for his extraordinary help and support in my project. Apart from that I would like to thanks him for supporting and inspiring me to work in physics despite not being a physics background student. I thank him for providing me this excellent platform that will act like a stepping stone for my future endeavors in physics too.

Mr. Bibekanandani Maji, research scholar, IITB, for helping me out at various instants in my project, especially in things related to *magneto-caloric* effect and in data of the compound that I studied.

My friends and KVPY invitees, for their support and never letting me feel that I was away from home and my family for extending support all the way, and uncompromised support for such a unique experience of a lifetime.

Thank you everyone.

Model

What is a model?

In the most general sense, a model is anything used in any way to represent anything else, to study, predict or simply for illustrative purposes. Some models are physical objects, for instance, a model of a building which may be assembled, and may even be made to work like a real building it represents. In general these models are used to help us know and understand the subject matter they represent. The term conceptual model may be used to refer to models which are represented by concepts or related concepts which are formed after a *conceptualization process* in the mind.

Conceptual models (models that are conceptual) range in type from the more concrete, such as the mental image of a familiar physical object, to the formal generality and abstractness of mathematical models which do not appear to the mind as an image.

There are a large class of modes, all used to simplify details of something complex and study its nature better. Modelling is an essential and inseparable part of all scientific activity, and many scientific disciplines have their own ideas about specific types of modelling. There are a wide class of models that analysts build which include mental models, logical models, *mathematical models* and *scientific models*.

Why to model?

Although the following article will also try to build a base of usefulness of models, here I have given a very general insight of why do we make models. In the most general sense, models are typically used when it is either impossible or impractical to create experimental conditions in which scientists can directly measure outcomes. Direct measurement of outcomes where possible will always be more reliable than modelled estimates of outcomes. When predicting outcomes, models use assumptions, while measurements do not. However, it is important to note that in analysing the data collected from measurements, assumptions are made albeit different to those made through the use of a model. As the number of assumptions in a model increases, the accuracy and relevance of the model will likely diminish.

Mathematical modelling

Mathematical modelling is the process of building conceptual models, in the language of mathematics. Mathematical models can take many forms, including but not limited to dynamical systems, statistical models, differential equations, or game theoretic models. There isn't a fixed boundary in different types of models and these models can overlap, involving a variety of *abstract structures* which are models themselves. In general, mathematical models may include logical models as well, as far as logic is taken as a part of mathematics.

A related term physical modelling involves a process of making prediction about physical processes and explaining their behaviour. It should not be confused with physical models which are entirely a different thing. The physical process under study commonly involves a physical system which undergoes that physical process. A physical system is something that *in the real world* we are interested in studying. As my whole study is cantered about mathematical

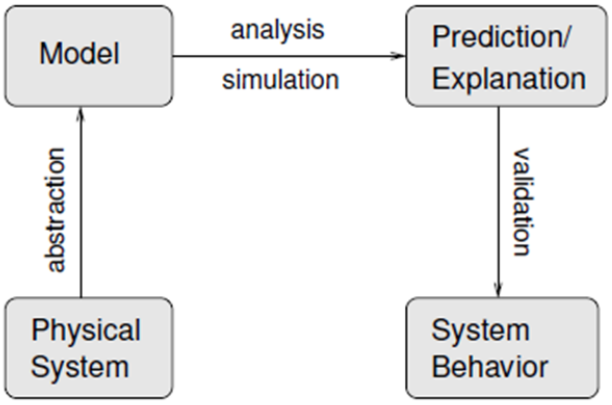
modelling of physical systems in computers so I would like to stress a bit on what exactly mathematical models are and how they are used.

When we pick a particular real world scenario for physical modelling, the very first step is to identify sufficient and useful abstractions that are both sufficient and necessary to model the required behaviour of that particular real world situation. Using these abstractions we define a *mathematical model*.

Using these abstractions we build a model which is a simplified but is a profound description of the physical system. It is simplified in the sense that only the required details are taken into account that are necessary for describing a particular behaviour of the system while neglecting others irrelevant things while at the same time profound and precise in the sense it is stated in the language of mathematics and involves *mathematical rigor*.

Indeed the very process of building a model is called abstraction. In this context, *"abstract"* is the opposite of *"realistic"* so an abstract model bears little direct resemblance to the physical system it models, in the same way that abstract art does not directly depict objects in the real world. A realistic model is one that includes more details and corresponds more directly to the real world. Abstraction involves making justified decisions about which factors to include in the model and which factors can be simplified or ignored.

For example in physics we usually have these mathematical models that are used to express physical theories precisely. The mathematics involved in modelling physics is usually much more complicated. So it is common to sue idealized models in physics to simplify things. In that context we only take some useful abstractions while ignore others. Massless rope, point particles, ideal gases and the particle in a box are some examples of simplified models used in physics,



This is the block diagram depicting the typical steps in modelling a physical system

So a model can have varied amount of resemblance to real world scenario depending on the abstractions that we choose to define the model. All models are in *simulacra*, that is, simplified reflections of reality, but, despite their inherent falsity, they are nevertheless extremely useful. Although a more realistic model explains more properties of the real physical system and is more likely to make better and accurate predictions of the physical system but it is not necessarily better. Models are useful because they can be analysed mathematically and simulated computationally. Models that are too realistic might be difficult to simulate and impossible to analyse. In that sense ideally, our goal should be to define a model that closely resembles the physical system but at the same time simple enough to be well

understood and is useful. A particular example of a complicated model that however has undoubtedly made excellent predictions of nature is quantum mechanics. A noted physicist, Mr Feynman said that it's safe to say that nobody understands quantum mechanics.

In general physicist tries to make models which are more closely related to real world that can explain every aspect of that phenomenon and can make accurate predictions. One such endeavour widely under research is *'theory of everything'*, that aims to explain all types of forces in nature, all in one single mathematical model. However in other disciplines like social sciences, mathematical biology involving population dynamics, it's safe to make a model that serves and explain its purpose well and not that rigorous. So emphasis is on simplicity rather that rigor.

A model is successful if it is good enough for its purpose. If we want to study the projectile of a stone, considering it a point object would be a good approximation. This will simplify all the irrelevant forces that we would have to consider otherwise and motion will still remain the same. The required analysis on projectile motion can be performed on the model without any error. On the other hand if you are a physicist and want to explain what point of stone is going to land on earth, you need more work.

Checking whether a model is good enough is called validation. The strongest form of validation is to make a measurement of an actual physical system and compare it to the prediction of a model. If that is infeasible, there are weaker forms of validation. One is to compare multiple models of the same system. If they are inconsistent, that is an indication that (at least) one of them is wrong, and the size of the discrepancy is a hint about the reliability of their predictions.

In practise a model is evaluated first and foremost by its consistency to empirical data; any model inconsistent with reproducible observations must be modified or rejected. One way to modify the model is by restricting the domain over which it is credited with having high validity. A case in point is Newtonian physics, which is highly useful except for the very small, the very fast, and the very massive phenomena of the universe. However, a fit to empirical data alone is not sufficient for a model to be accepted as valid. Other factors important in evaluating a model include:

- Ability to explain past observations
- Ability to predict future observations
- Cost of use, especially in combination with other models
- Refutability, enabling estimation of the degree of confidence in the model
- Simplicity, or even aesthetic appeal

Computer simulations

Simulation is the imitation of some real thing, state of affairs, or process. The act of simulating something generally entails representing *certain key characteristics* or *behaviours* of a selected physical or abstract system. In general meaning of simulation is much broader and depends on the particular context.

In the most general sense, we create a model of the system and simulate that model for studying certain system dynamics and characteristics. The use of simulations enables its users to draw conclusions about the real system by studying and analysing the model.

The major reasons for developing a model for simulation, as opposed to analysing the real system, include economics, unavailability of a "real" system, and the goal of achieving a deeper understanding of the relationships between the elements of the system.

To study the system completely, we have to perform various types of simulations on the model. A *steady state simulation* provides information about the system at a specific instant in time (usually at equilibrium, if such a state exists). A *dynamic simulation* provides information over time. A simulation brings a model to life and shows how a particular object or phenomenon will behave. These two types of simulations are generally conducted in electrical engineering on circuit models.

Apart from use of simulations in scientific research contexts to analyse and interpret data, a different version of simulations are used in task or situational training areas in order to allow humans to anticipate certain situations and be able to react properly or decision-making environments to test and select alternatives based on some criteria.

With simulation a decision maker can try out new designs, layouts, software programs, and systems before committing resources to their acquisition or implementation; test why certain phenomena occur in the operations of the system under consideration; compress and expand time; gain insight about which variables are most important to performance and how these variables interact; identify bottlenecks in material, information, and product flow; better understand how the system really operates (as opposed to how everyone thinks it operates); and compare alternatives and reduce the risks of decisions.

Traditionally, the formal modelling of systems has been via a mathematical model, which attempts to find *analytical solutions* enabling the prediction of the behaviour of the system from a set of parameters and initial conditions. Computer simulation is often used as an adjunct to, or substitution for, modelling systems for which simple closed form analytic solutions are not possible. I have done similar work in this project in Physics department. Although there are many different types of computer simulation; the common feature they all share is the attempt to generate a sample of representative scenarios for a model in which a complete enumeration of all possible states would be prohibitive or impossible.

Environment for computer simulations

Recently, in the last two or so decades, due to widespread availability of personal computer that can perform great amount of calculations in no time, mathematical modelling and related simulations is increasingly being used in various disciplines to enhance the success rate and decrease cost and for scientific and academic affairs. As explained in some examples above, mathematical modelling and related simulations is no longer limited to natural sciences (like Physics, earth science, meteorology and biology) and engineering disciplines (e.g. – computer science and artificial intelligence), but also in social sciences (such as economics, psychology, sociology and political science); physicists, engineers, statisticians, operations research analysts and economists use mathematical models most extensively.

Two good tools among many others, which are equally good and popular among scientist, industrials and computer enthusiasts, are Matlab and pyLab. Although both are equally good at their own places, easy and have wide online

support and user base but I think pyLab is a bit more inclined towards coding. On the other hand Matlab is a relatively easy to get started and is popular among those who are not related to computer science and coding in particular. My final goal was to simulate thing for physics department, where people are not coders, so I choose Matlab as my development environment.

Static models

A static model is one which does not account for the element of time and its mathematical form does not contain differential or difference equations of time. This is in contrast to dynamic model which does account for time and use differential equations to model changes in a system in response to usually changing input signals.

Equations of static models define relationships between elements of the modelled system which are not during runtime and attempt to find a state in which the system is in equilibrium. Static models are usually defined as structural in contrast to dynamical models which are said to represent behaviour of static components of the system.

Numerical Methods

Introduction

The best way I thought to describe numerical methods is what I have seen in Wikipedia, which says numerical analysis is the study of algorithms that use numerical approximation (as opposed to general symbolic manipulations) for the problems of mathematical analysis

In general through the use of numerical methods many problems can be solved that would otherwise be thought to be insoluble. So it forms a very important and fundamental part of solving problems related to modelling in computers. Typically we encounter problems in nature that are often modelled by equations that are impossible or very difficult to solve analytically. So these numerical methods give us ways to solve such equations and find approximated but practically acceptable answers. So in this section I start with discussing a bit about common numerical methods and tried to visualize their working in matlab. I intentionally leave to describe these methods and have given more emphasis on visual internal working of these methods.

Common Code

```
% drawfunc.m
% This code draws the polynomial function
function[] = drawline(a,b)
x=linspace(a,b,1000);
P=[1 -5 6];
y=polyval(P,x);
plot(x,y,'-r','LineWidth',2)
hold on
%R=roots(P);
%plot(R,[-0.5:0.001:2 -0.5:0.001:2])
```

```
% drawline.m
% This function draws the line
function [] =drawline(a,fa,b,fb)
x=linspace(a,b,1000);
m=(fb-fa)/(b-a);
y=fa+m.*(x-a);
plot(x,y,'-b')
```

Solutions of nonlinear equations

I would like to explain the problem in the most concise form. The problem is that if we have one equation in one unknown, and we need to find one, some, or all of the values of that unknown which satisfy the equation. That is, we have to solve f (x) = 0, where f (x) is a known function of x. This is finding the value of x where the graph of y = f (x) intersects the x axis. There may be more than one solution.

Secant Method

The secant method is a root-finding algorithm that uses a succession of roots of secant lines to better approximate a root of a function. This method is generally applied when the function whose roots are to found is not easy to differentiate or non-differentiable. We need two points or starting values $(x_0,f(x_0))$ and $(x_1,f(x_1))$ and the method continues for x_2 as

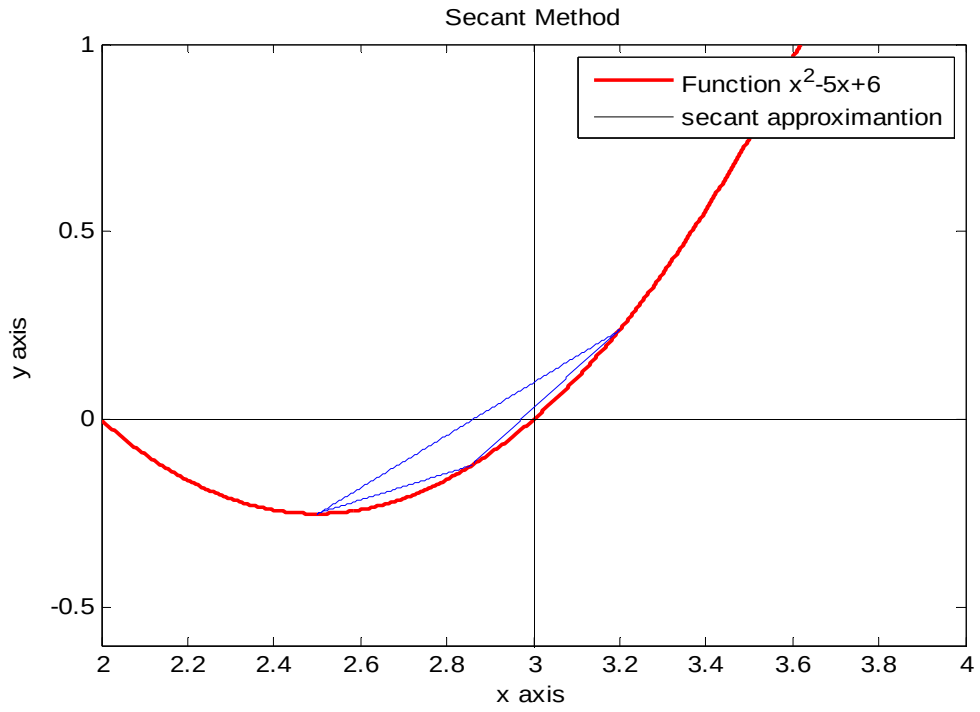
$$x_{n+1} = x_n + \delta_n \quad \text{where } \delta_n = \frac{-f(x_n)\delta_{n-1}}{f(x_n) - f(x_{n-1})}.$$

Matlab Code

```
% Initial values
a=3.2;
b=2.5;

P=[1 -5 6];
figure;
drawfunc(1.6,4);    % Draws the function
hold on;
```

```
% Draw axis
p_y=-1:0.1:2;
p_x=3*ones(length(p_y));
plot(p_x,p_y,'k');

p_x=2:0.1:4;
p_y=zeros(length(p_x));
plot(p_x,p_y,'k');
xlim([2 4]);
ylim([-0.6 1]);
title('Secant Method');
xlabel ('x axis');
ylabel ('y axis');
legend ('Function x^2-5x+6','secant approximantion');

%algorithm begins
x0=a; x=b;
f0=polyval(P,x0);
f=polyval(P,x);
drawline(x0,f0,x,f);
pause

for n=1:100
   f0=polyval(P,x0);
   f=polyval(P,x);

   x1=x0-((f0*(x-x0))/(f-f0)) ;
   f1=polyval(P,x1);
       drawline(x,f,x1,f1);
   x0=x;
   x=x1;
   pause
end
```

The result



Regula falsi

Regula falsi is often described using a *regula falsi theorem* which is, assume that $f \in C[a, b]$ and that there exists a number $r \in [a, b]$ such that $f(r) = 0$.

If $f(a)$ and $f(b)$ have opposite signs, and

$$c_n = \frac{a_n f(b_n) - b_n f(a_n)}{f(b_n) - f(a_n)}$$

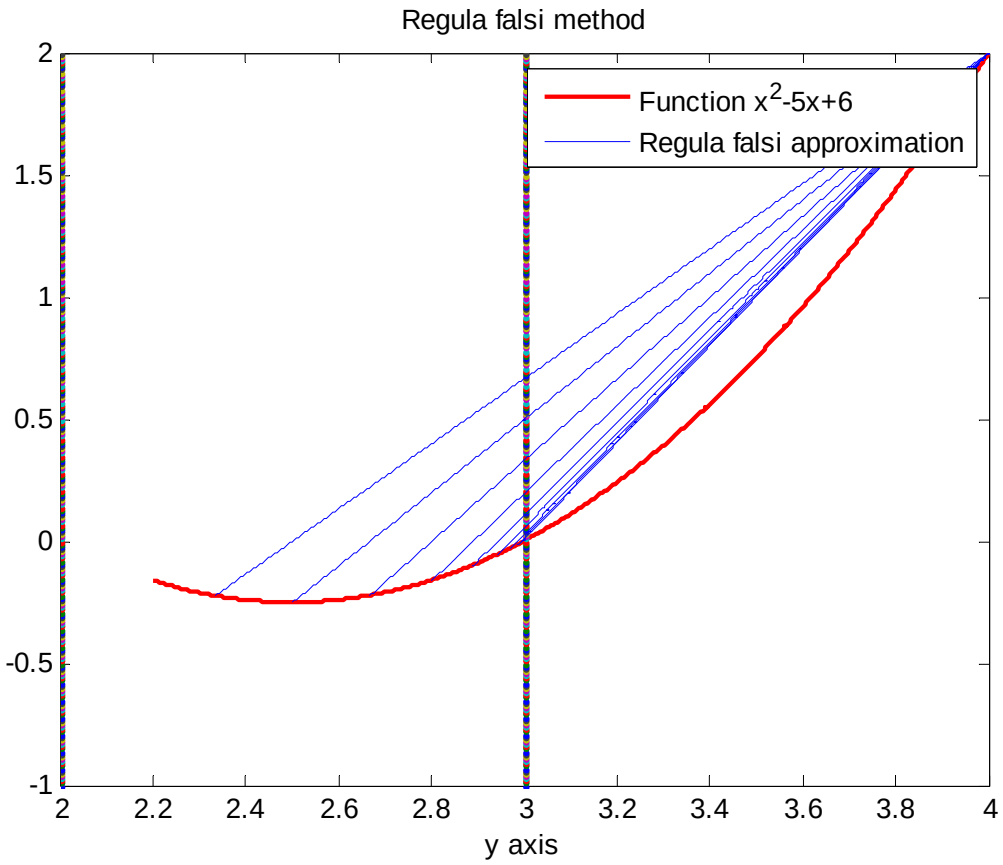
Represents the sequence of points generated by the Regula Falsi process, then the sequence $\{c_n\}$ converges to the zero $x = r$.

That is $\lim_{k \to \infty} c_n = r$

Matlab Code

```
% Initial parameters
a=2.2
 b=4
% xn+1=xn- f(b)-f(a)
P=[1 -5 6];
figure;
drawfunc(a,b);
hold on;
plot(3,-1:0.01:2)
plot(2,-1:0.01:2)

title('Regula falsi method');
xlabel('x axis');
xlabel('y axis');
legend('Function x^2-5x+6','Regula falsi approxima-
tion');

while 1
    fb=polyval(P,b);
    fa=polyval(P,a);
    w=(fb*a-fa*b)/(fb-fa);
    fw=polyval(P,w);
    if fa*fw<0
        % line should be b/w a and w
        drawline(a,fa,w,fw);
        b=w;
    else
        drawline(b,fb,w,fw);
        a=w;
    end
    pause;

end
```

The result



Newton raphson Method

Newton raphson method is the most commonly used method for calculating the roots of the function f(x). It converges very fast as compared to other root finding methods and is fairly simple to implement.

Given a function $f(x)$ and its derivative $f'(x)$, we begin with a first guess x_0 for a root of the function. Provided the function is reasonably well-behaved a better approximation x_1 is

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}.$$

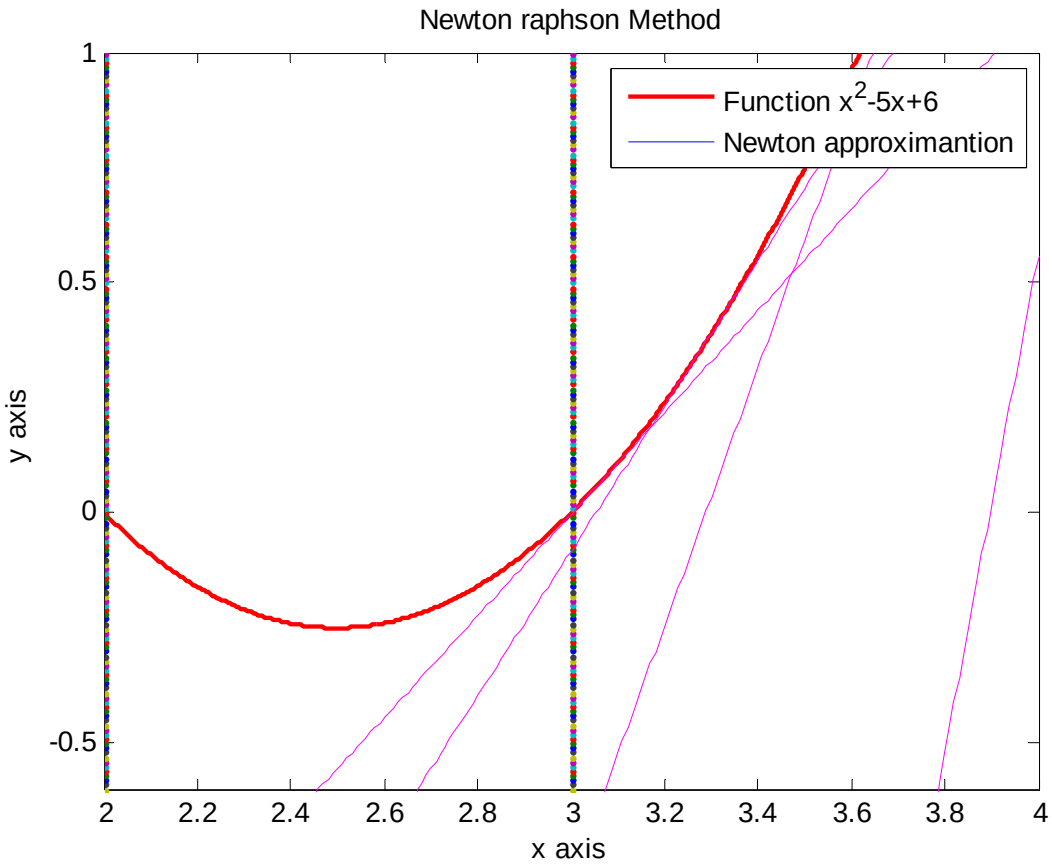
Geometrically, x_1 is the intersection with the x-axis of a line tangent to f at $f(x_0)$. The process is repeated until a sufficiently accurate value is reached:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}.$$

Matlab Code

```
% This program finds roots of a function numerically
using newton raphson's method
a=3.2;
b=2.5;

P=[1 -5 6];
dP=[2 -5];
figure;
drawfunc(1.6,4);
hold on;
plot(3,-1:0.01:2);
plot(2,-1:0.01:2);
xlim([2 4]);
ylim([-0.6 1]);

title('Newton raphson Method');
xlabel ('x axis');
ylabel ('y axis');
legend ('Function x^2-5x+6','Newton approximantion');

%algorithm begins
x0=a+2;
f0=polyval(P,x0);

xr=2:0.01:4;


for n=1:100
   f0=polyval(P,x0);
   df=polyval(dP,x0);


   x1=x0-(f0/df);

   yr=(df*xr)+f0-(df*x0);
   plot(xr,yr,'-m');

   x0=x1;
   pause
end
```

The result



Magnetic field of an infinite line wire carrying current

Magnetic field of a current carrying wire is given at a point by biot-savart law, which is an equation in electromagnetism that describes the magnetic field B generated by an electric current. This law is stated mathematically as follows

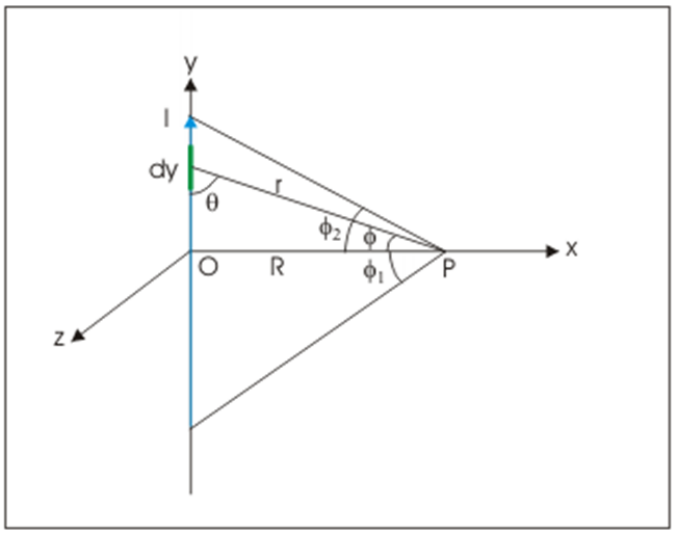
$$\mathbf{B} = \int \frac{\mu_0}{4\pi} \frac{I d\mathbf{l} \times \hat{\mathbf{r}}}{|r|^2},$$

Where;

- *I* is the current,

- $d\mathbf{l}$ is a vector, whose magnitude is the length of the differential element of the wire, and whose direction is the direction of conventional current,
- $\mathbf{B}$ is the net magnetic field,
- μ_0 is the magnetic constant,
- $\hat{\mathbf{r}}$ is the displacement unit vector in the direction pointing from the wire element towards the point at which the field is being computed, and
- $\mathbf{r}=r\hat{\mathbf{r}}$ is the full displacement vector from the wire element to the point at which the field is being computed.

This law can be easily used to calculate the magnetic field calculated by an infinite as well as a finite length wire carrying current, by using the following relations, in relation to this figure



$$B = \frac{\mu_0 I}{4\pi R} \int_{\varphi_1}^{\varphi_2} I\cos\varphi \mathbb{d}\varphi$$

$$B = \frac{\mu_0 I}{4\pi R} \left(\sin\varphi_2 - \sin\varphi_1\right)$$

The later expression can be derived from the former by integrating the expression analytically. Although field can be calculated by using both, and probably easily using the later but in order to analyse the results of the matlab integrators, and in the essence of using fundamental equations, I used the former equation to obtain another equation that has the differential element dz instead of dw and integrate the complex equation numerically to obtain the field at each point in plane. The derivation process that I did is as follows.

$$\int dB = \frac{\mu_0 I}{4\pi R} \int_{\phi_1}^{\phi_2} I \cos\phi \, d\phi.$$

Position of wire $= (x_0, y_0)$

Length of wire $= (z_1, z_2)$

$(z_1, x_0, y_0) \rightarrow (z_2, x_0, y_0)$

$$\int dB(x) = \frac{\mu_0 i}{4\pi} \int_{l_1}^{l_2} \frac{dl \sin\theta}{r^2}$$

$$\int_{z_1}^{z_2} \frac{\sin\theta}{r^2} dz$$

$$\int_{z_1}^{z_2} \frac{z}{r^3} dz.$$

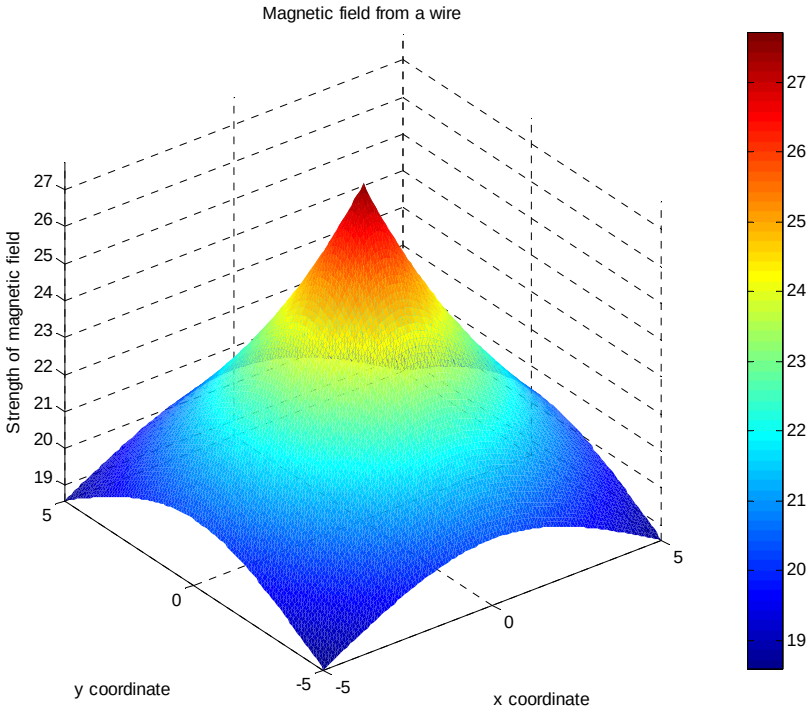
$$= \frac{\mu_0 i}{4\pi} \int_{z_1}^{z_2} \frac{z \, dz}{\left(\sqrt{(x-x_0)^2 + (y-y_0)^2 + z^2}\right)^3}$$

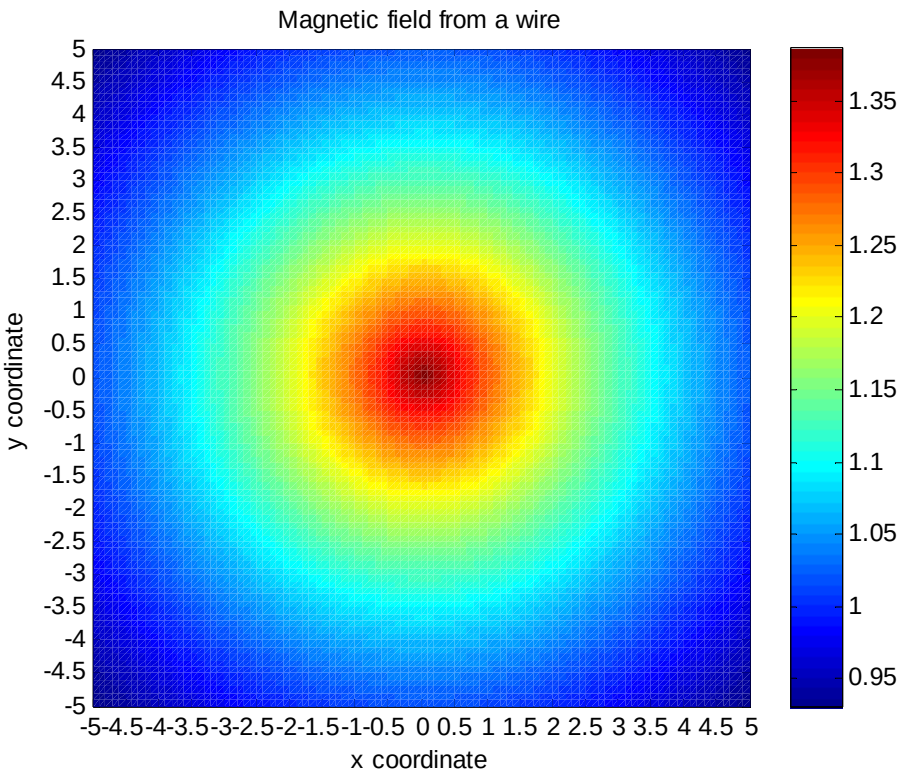
MATLAB CODE

```
close all;
clear all;
clc

% Given parameters
i=1;
z1 = 2;
z2 = 8;
x0 = 0;
y0 = 0;

% Draw the Line in the z direction
plot(0,0,'*','LineWidth',3);

% Make a matrix of points for computation
x = [-5:0.1:5];
y = [-5:0.1:5];
[X,Y]= meshgrid(x,y);

result = zeros(size(X));

% Constant Q = mu0*i/4pi
Q = i*1e-7;

% Perform integration for each such point
%result(:,:)=quad(@(z) z/(sqrt((X(:,:)-x0).^2 +
(Y(:,:)-y0).^2 ) + z.^2),z1,z2);

for n = 1:length(y)
    %row

    for k= 1:length(x)
        f = @(z) z./(sqrt((x(k)-x0).^2 + (y(n)-y0).^2 )
+ z.^2);
        result(n,k)=quad(f,z1,z2);
    end
end

%result = result./max(max(result)); % normalize the
result
```

```
% GRAPHICS
figure(1);
pcolor(X,Y,result);%.^0.1);
hold on;
colormap(jet)
shading interp;
axis equal;
xlabel('x coordinate');
ylabel('y coordinate');
xlim ([-5 5]);
ylim ([-5 5]);
set(gca,'XTick',[-5:5/10:5]);
set(gca,'YTick',[-5:5/10:5]);
title('Magnetic field from a wire')
colorbar

plot(0,0,'xb','LineWidth',2);



figure(2);
surf(X,Y,result*20);%.^0.1);
colormap(jet)
shading interp;
axis equal;
xlabel('x coordinate');
ylabel('y coordinate');
zlabel('Strength of magnetic field')
xlim ([-5 5]);
ylim ([-5 5]);
%set(gca,'XTick',[-5:5/10:5]);
%set(gca,'YTick',[-5:5/10:5]);
title('Magnetic field from a wire')
colorbar
```

The results


Magnetic field from a wire
Strength of magnetic field
27
26
25
24
23
22
21
20
19
5
0
-5
y coordinate
-5
0
5
x coordinate
27
26
25
24
23
22
21
20
19


Magnetic field from a wire
y coordinate
5
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-2.5
-3
-3.5
-4
-4.5
-5
-5 -4.5 -4 -3.5 -3 -2.5 -2 -1.5 -1 -0.5 0 0.5 1 1.5 2 2.5 3 3.5 4 4.5 5
x coordinate
1.35
1.3
1.25
1.2
1.15
1.1
1.05
1
0.95

Specific heat capacity

The equation that I'm supposed to plot is in the scanned image below. It explains the variation of specific heat capacity at low as well as high temperature. The decrease of specific heat capacity at low temperatures cannot be explained by Dulong-Petit law. The new equation that consist of phonon and electronic contribution and accounts for the specific heat capacity at all temperatures. Although this model is also unable to accurately explain the trends of specific heat capacity, it can still explains the decrease of specific heat capacity at low temperatures.

$$C = \gamma T + 9NR\left(\frac{T}{\Theta_D}\right)^3 \int_0^{\Theta_D/T} \frac{x^4 e^x}{(e^x-1)^2}\,dx$$

C → Specific heat capacity.
γ → 20 mJ/mol K^2
N → 6×10^{-3}/g
Θ_D → 350 K.
R → 8.314 (UGC)

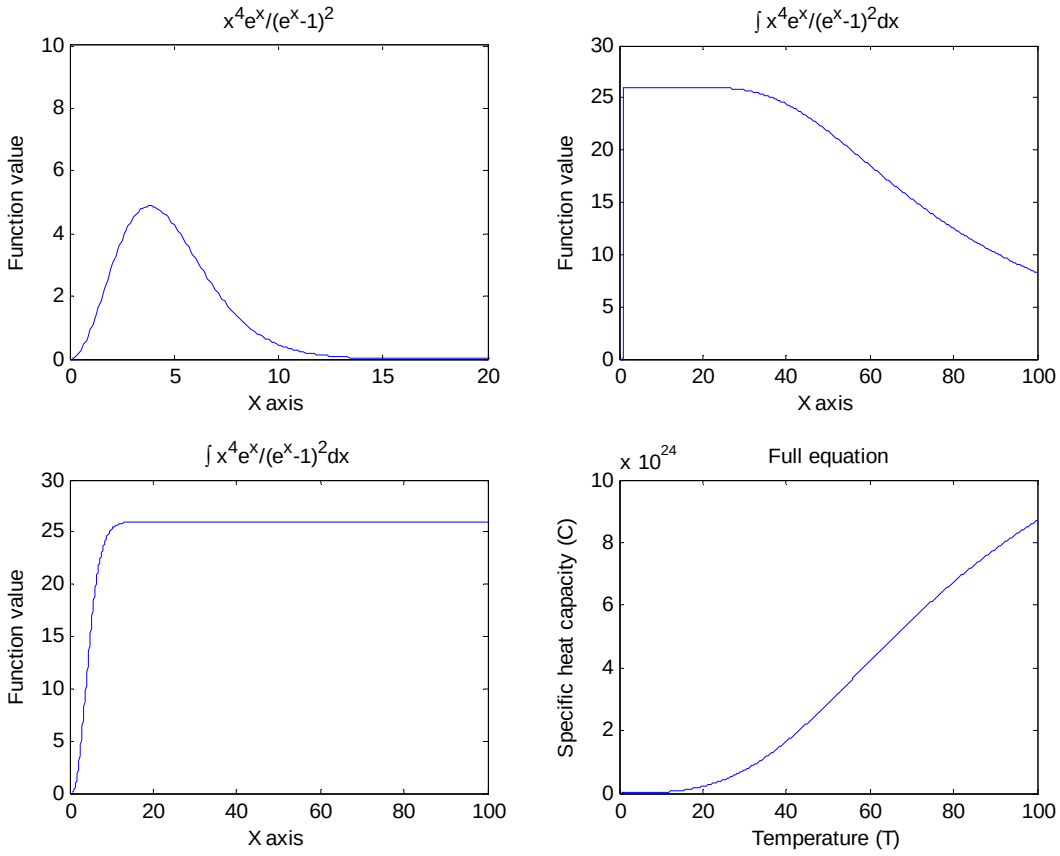
Matlab code

```
close all;
clear all;
clc

gama = 20 * 1e-3;
N = 6 * 1e23;
thetad = 350;
R = 8.314;

T = 0:0.1:600;
f = @(x) ((x.^4) .* exp(x)) ./((exp(x)-1).^2);

C = linspace (0,0,length(T));

figure
title('Integral part of the equation \int
{x^4e^x}/{(e^x-1)^2}dx')
```

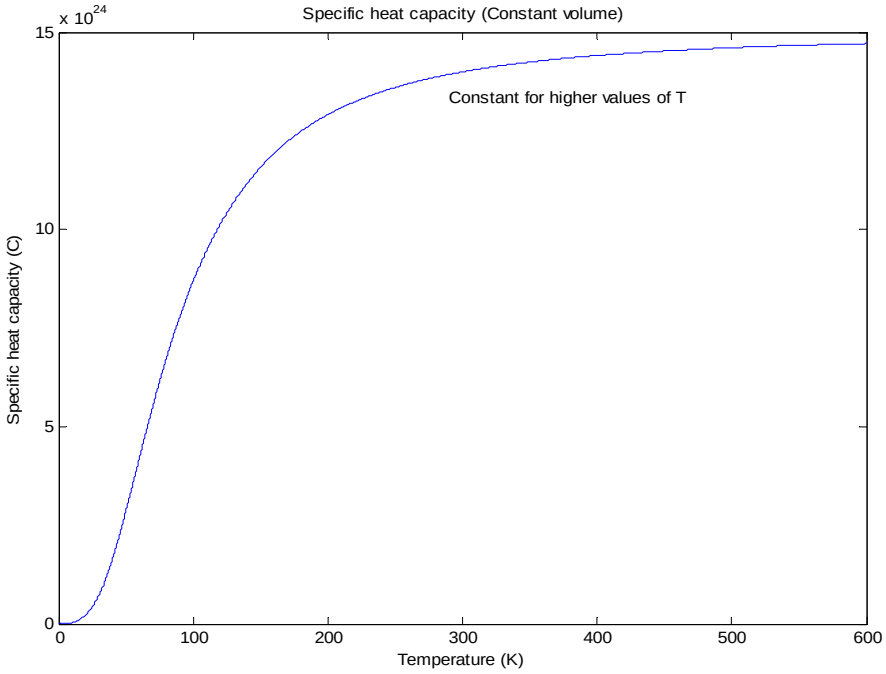
```
for n=1:length(T)
    C(n) = gama*T(n) +
9*N*R*((T(n)/thetad)^3)*quad(f,0,thetad/T(n));
end

plot (T,C);
title('Specific heat capacity (Constant volume)');
title(texlabel(f,'literal'))
ylabel('Specific heat capacity (C)');
xlabel ('Temperature (K)');
gtext('Constant for higher values of T')
```

Result

Study of function in the integral



The overall equation

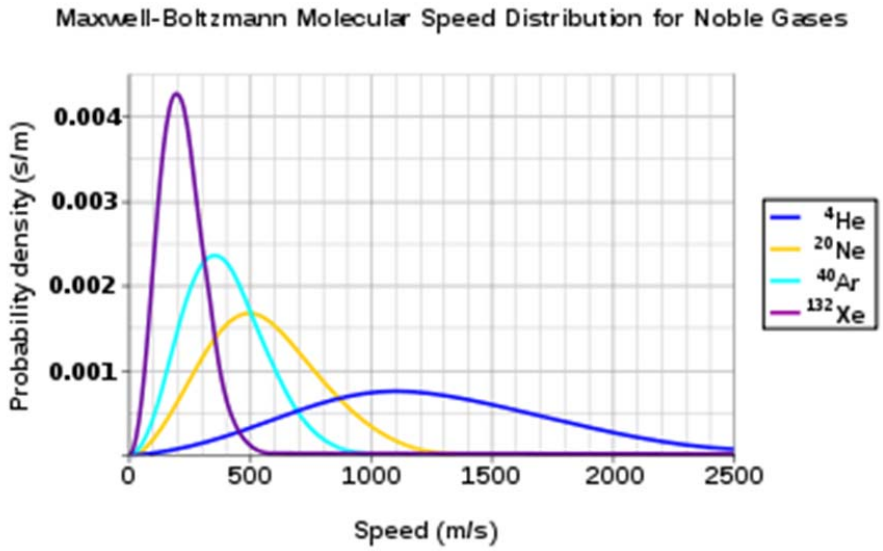


The overall variation of the specific heat capacity with temperature is shown in the figure above. The expression of specific heat capacity is a general one that tends to explain the characteristic curve of specific heat capacity at low temperatures as well as at high temperatures. The expression which consists of phonon and electronic contribution seems to be consistent with Dulong-petit law. Dulong petit law says that specific heat capacity per molar mass is a constant value, equal to 3R. This result can be observed in the figure above at high temperatures, where the curve tends to be flat and tends to assume a constant value, as predicted by Dulong-petit law.

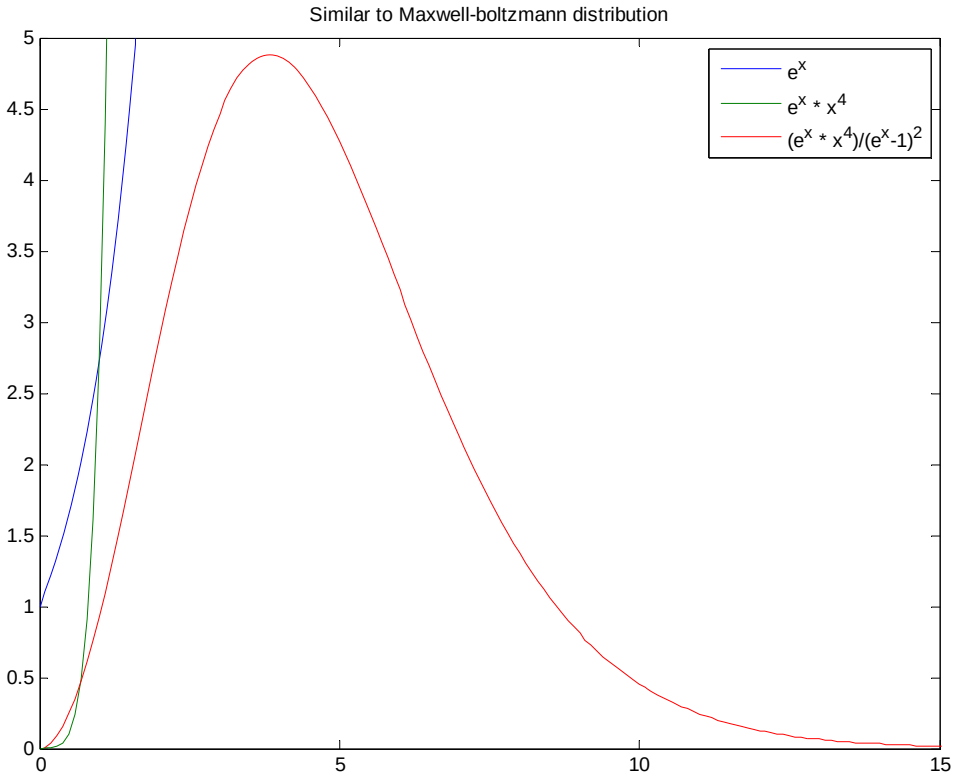
One more thing that I found and felt like worth mentioning is that the function in the integral is similar to Maxwell-Boltzmann distribution which relates the distribution of velocity of molecules in a gas.

Maxwell- Boltzmann distribution

$$f(v) = \sqrt{\frac{2}{\pi}\left(\frac{m}{kT}\right)^3} v^2 \exp\left(\frac{-mv^2}{2kT}\right)$$



The curve of the integral of function f, is similar to Maxwell Boltzmann distribution, in the above image.

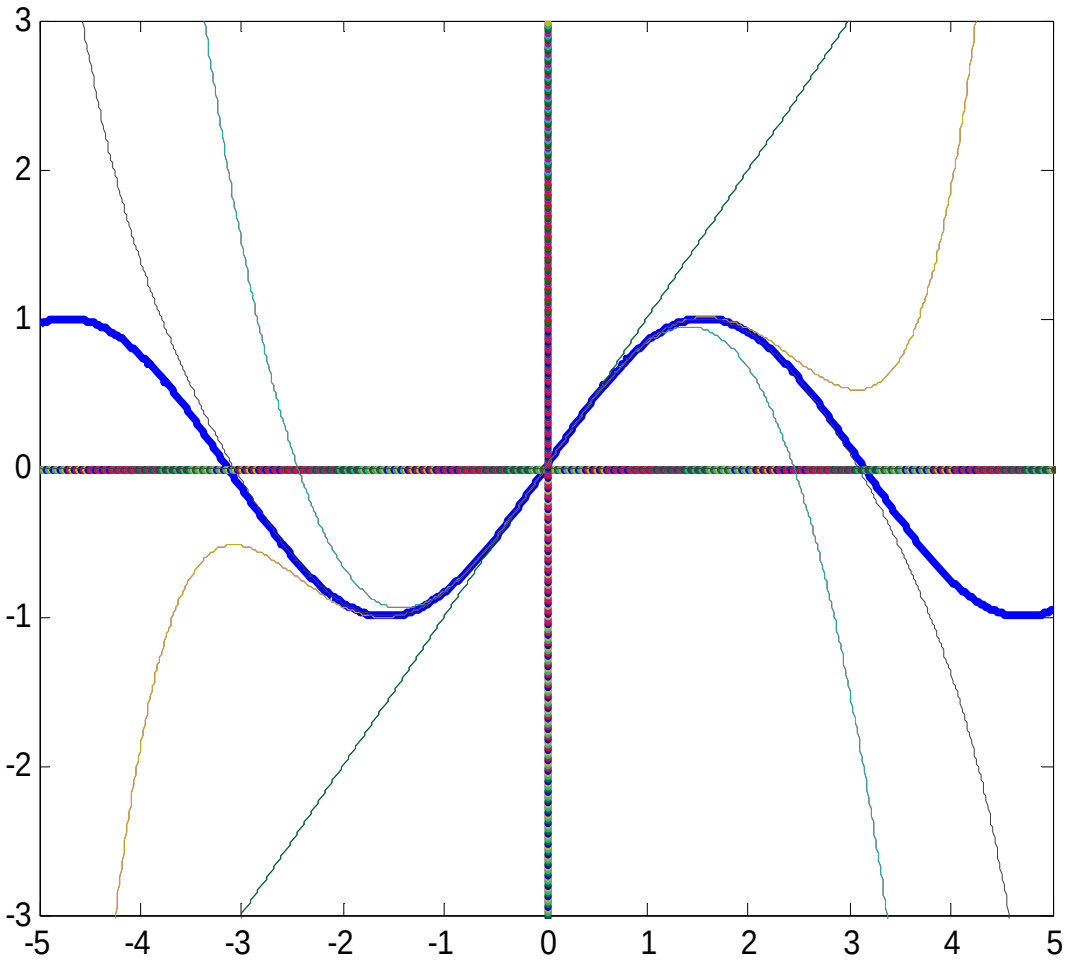


Taylor polynomials

Taylor series is a **representation of a function** as an **infinite sum of terms** that are calculated from the function's derivatives at a single point. If Taylor series is cantered at 0, it is known as *maclaurin series*.

It is a common practise to approximate a function by using a finite number of terms of its Taylor series. Taylor theorem gives quantitative estimates of the error of its approximation. A finite number of initial terms of the Taylor series of a function are called Taylor polynomial. The Taylor series of that function is the limit of that Taylor polynomial provided that limit exist. The more you increase the degree of the Taylor's polynomial, the more it hugs the curve.

The result



Matlab code

```
% This script illustrates how taylor polynomials of
higher degree, gets
% closer to the function and better approximates it.

x = linspace(-5,5,1000);

%define function to find derivative
syms t;
g=sin(t);

y=subs(g,t,x);

figure;
plot(x,y,'-','LineWIdth',2.5);
hold all;

plot(x,0);
plot(0,-3:0.01:3)

xlim([-5 5]);
ylim([-3 3]);

% Find the derivatives
dg = diff(g,t);
dydx = subs(dg,t,x);


% Fst order
P = [subs(g,t,0)];
Px=polyval(P,x);
plot(x,Px);

% Draws taylor polynomials
for n=1:100
    P = [subs(dg,t,0)/factorial(n), P];
    Px=polyval(P,x);
    plot(x,Px)
    pause;
    dg = diff(g,t,n+1);
    dydx = subs(dg,t,x);
end
```

Dynamical system

Introduction

As already explained in the section static models, dynamical modelling is used model systems during runtime and represent changes in states during the course of time for a system. Usually there are differential equations that models changes in system in response to changing input signals.

Dynamical modelling is used in particular to model dynamical systems. A dynamical system is a concept in mathematics where a fixed rule describes the time dependence of a point in a geometrical space. At any given time dynamical system has a state given by a set of real numbers (a vector) that can be represented by a point in appropriate state space (a geometrical manifold). Generally small changes in the state of the system create small changes in the numbers. The so called *evolution rule* of the dynamical system is a fixed rule that describes what future states follow from the current state. The rule is, as we say ***deterministic***; in other words, for a given time interval only one future state follows from the current state.

The concept of a dynamical system has its origins in Newtonian mechanics. There, as in other natural sciences and engineering disciplines, the evolution rule of dynamical systems is given implicitly by a relation that gives the state of the system only a short time into the future. (The relation is either a differential equation, difference equation or other time scale.) To determine the state for all future times requires iterating the relation many times—each advancing time a small step. The iteration procedure is referred to as solving the system or integrating the system. Once the system can be solved, given an initial point it is possible to determine all its future positions, a collection of points known as a *trajectory* or *orbit*.

In particular, dynamical systems are modelled using lumped-parameters and high fidelity mathematical models given in the form of nonlinear differential equations (ordinary and partial) and difference equations. These equations are often not possible to solve analytically, so solved using numerical method implemented in computers. Following are some examples of modes of dynamical systems in matlab

Ball throw

This is the simplest dynamical system, in which I have simulated the projectile motion of a ball from a certain height. I have plotted and analysed the results using two methods, one among them is a approximated taylor like numerical solution that produce erroneous results as compared to accurate analytical method.

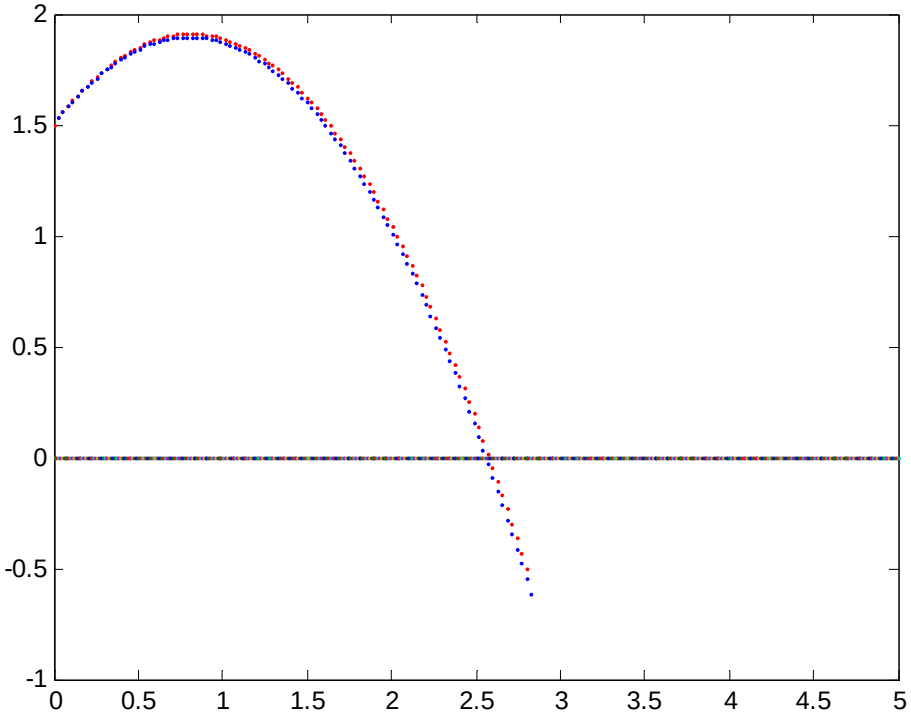
The ball throw can be modelled using the following differential equations which can be easily modelled in matlab.

$$\dot{V}_x = a_x = -V_x k/m$$
$$\dot{V}_y = a_y = -V_y k/m - g$$

Matlab Code

```
% script that simulates the throw of a ball from a
height

% some constants
h0=1.5;
g=9.8;
v0=4;
theta = 45;

% make a time vector that divide the time interval into
1000 equally space intervals
t=0;
dt=0.01;

figure
plot(0:0.01:5,0);
hold on;

for n=1:100
    x = v0 * cos((pi/180)*theta)*t;
    y = h0 + ((v0 * sin((pi/180)*theta)).*t)-
((1/2)*g*(t.*t));
    t=t+dt;
    plot(x,y,'r-');
end

% now using the quantisation error technique
x=0;
y=h0;

vy=v0* sin((pi/180)*theta);

for n = 1:100
    dx = v0 * cos((pi/180)*theta)*dt;
    x=x+dx;

    vy = vy  - g*dt;
    dy=vy*dt;
    y=y+dy;
    plot(x,y)
end
```

Result



In the second method we have a constant dv = -gdt but when this term add to vy we have vy-gdt which is varying as time changes. So this value can add up to large value as time proceeds and so the error in the taylor method will increase because dy =vydt…

Van der pol's equation

Vander pol's equation is commonly used to demonstrate solving ODEs in matlab. This equation is a second-order nonlinear differential equation which may be expressed as two first-order equations as follows.

$$dx_1/dt = x_2$$

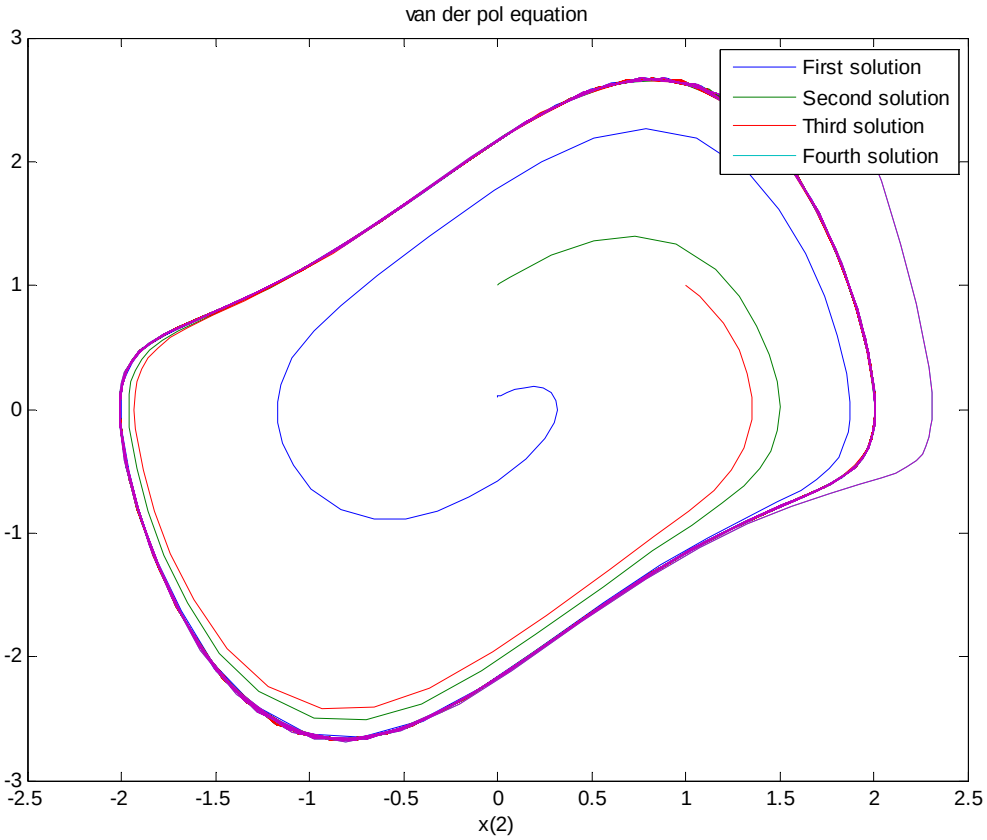
$$dx_2/dt = \epsilon(1 - x_1^2)x_2 - b^2x_1.$$

It can be solved using ode23 matlab solver and the solution of these first order equation has a stable limit cycle, which means if you plot the phase trajectory of the solution (the plot of x_1 against x_2) starting at any point in the positive x_1-x_2 plane, it always moves continuously into the same closed loop.

Matlab code

```
function rs = vander_de(t,X)
b = 1;

% e can range from 0.01 and 1.0
e = 1; rs = zeros(2,1);
rs(1) = X(2);
rs(2) = (e.*(1-(X(1).*X(1))) .* X(2)) - (b.*b .* X(1));


clear all;
%close all;
clc

[t,x] = ode23(@vander_de,[0,100],[0 0.1]);
plot(x(:,1),x(:,2))
hold all;

title('van der pol equation');
xlabel('x(1)');
xlabel('x(2)');

legend('First solution','Second solution','Third solu-
tion','Fourth solution');
```

The result



This is the phase plot of van der pol equation. The phase plot signifies the various states that the system can assume and follow. The path taken by the states of the system is called its trajectory.

Using b = 1 and e = 1 and starting with various initial values of x_1 and x_2, we can observe that all solutions, starting with various initial values converge and overlap into the same closed periodic loop.

Particle in magnetic field

Totally analytical solution is used to simulate the results. The analytical solution is of a particle moving with a velocity v in a magnetic field B , having a mass m and charge q. The motion is a circular in 2d plane and cycloid in 3d plane. These are the equations that govern the motion of the particle.

2d analytical solution

$$qvB = mv$$

$$r = \frac{mv}{qB}$$

$$\omega = \frac{|v|}{r}$$

$$x = x_0 + r\cos(\omega t)$$

$$y = y_0 + r\sin(\omega t)$$

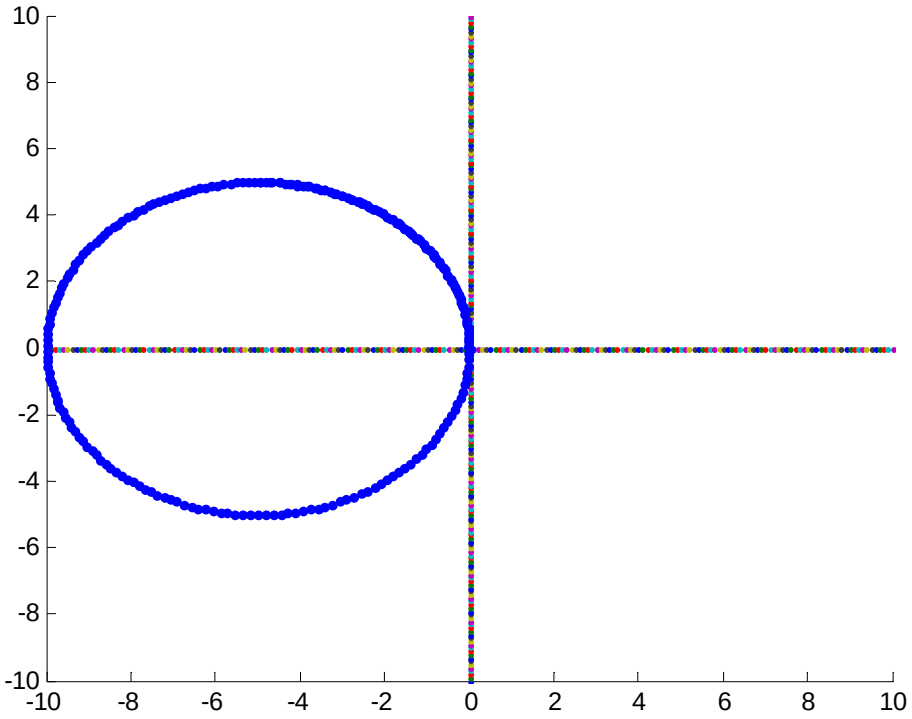
$$z = v_{\parallel} t$$

Matlab Code

```
v = [0 5 0];   %initial velocity
B = [0 0 -5];    %magnitude of B
m = 5;           % mass
q = 1;           % charge on particle
r0 = [0 0 0];    % initial position of particle
t = 0;

% find the circles centre
r = m*(norm(v))/(q*norm(B));
%theta = acos(dot(v,cross(v,B)));
theta = atan(v(2)/v(1))+pi/2;
xc=r0(1)+r*cos(theta);
yc=r0(2)+r*sin(theta);
w= norm(v)/r;

figure
hold on;
plot(-10:0.1:10,0);
plot(0,-10:0.1:10);
```

```
xlim([-10 10])
ylim([-10 10])
t=0;
%dt=0.01;
tic
for n=1:4000
    dt = toc
    tic
    x=xc+r*cos(w.*t + pi+theta);
    y=yc+r*sin(w.*t + pi+theta);
    plot(x,y,'.');
    t=t+dt;
    pause(0.00000000005)
end
```

The result



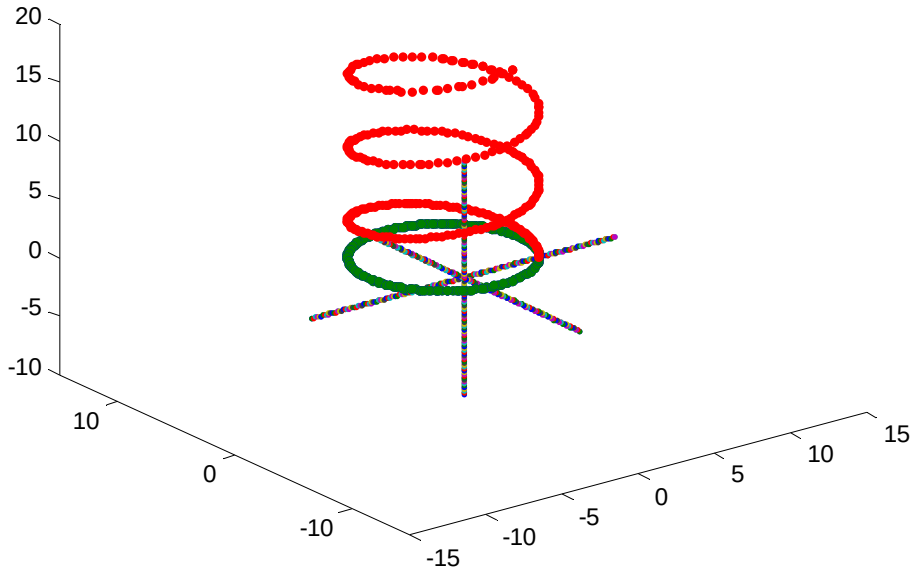
3d analytical solution

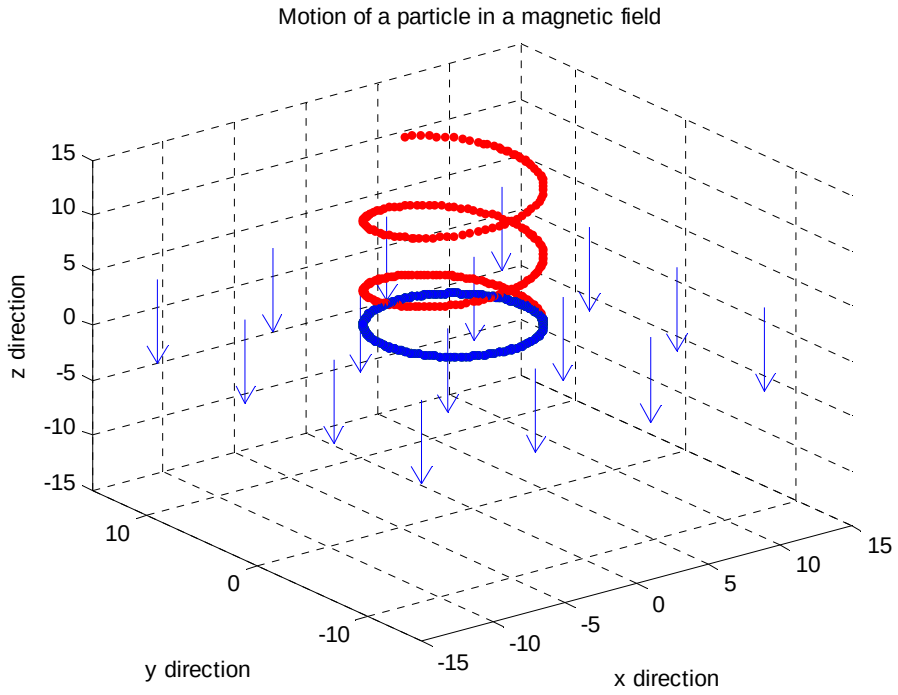
Matlab Code

```
% script that simulates a moving particle with some in-
itial velocity in a
% magnetic field B

v = [3 4 1];   %initial velocity
B = [0 0 -5];    %magnitude of B
m = 5;           % mass
q = 1;           % charge on particle
r0 = [5 0 0];    % initial position of particle
t = 0;

%find velocity parallel to B and perpendicular to B
v_para = (dot(v,B)/norm(B))*(B/norm(B));
v_per = v-v_para;

% find the circles centre
r = m*(norm(v_per))/(q*norm(B));

theta = atan(v_per(2)/v_per(1))+pi/2;

xc=r0(1)+r*cos(theta);
yc=r0(2)+r*sin(theta);

w= norm(v_per)/r;

figure
%to show B's direction
[x_q,y_q] = meshgrid(-15:8:15,-15:8:15);
z_q=10*ones(size(x_q));
u_q=zeros(size(x_q));
v_q=zeros(size(x_q));
w_q=-6*ones(size(x_q));
quiver3(x_q,y_q,z_q,u_q,v_q,w_q);
hold all;

% To draw axis and all
plot3(-15:0.1:15,0,0);
hold on;
plot3(0,-15:0.1:15,0);
plot3(0,0,-15:0.1:15)
```

```
xlim([-15 15])
ylim([-15 15])
%zlim([-15 15])

%legend
xlabel ('x direction');
ylabel ('y direction');
zlabel ('z direction');
title('Motion of a particle in a magnetic field');

t=0;
%dt=0.01;
tic
for n=1:4000
    dt = toc;
    tic
    x=xc+r*cos(w.*t + pi+theta);
    y=yc+r*sin(w.*t + pi+theta);
    z=v_para*t;
    plot3(x,y,z,'--.');
    hold on
    t=t+dt;
    pause(0.00000000005)
end
```

Results


20
15
10
5
0
-5
-10
10
0
-10
-15
-10
-5
0
5
10
15


Motion of a particle in a magnetic field
z direction
y direction
x direction
15
10
5
0
-5
-10
-15
10
0
-10
-15
-10
-5
0
5
10
15

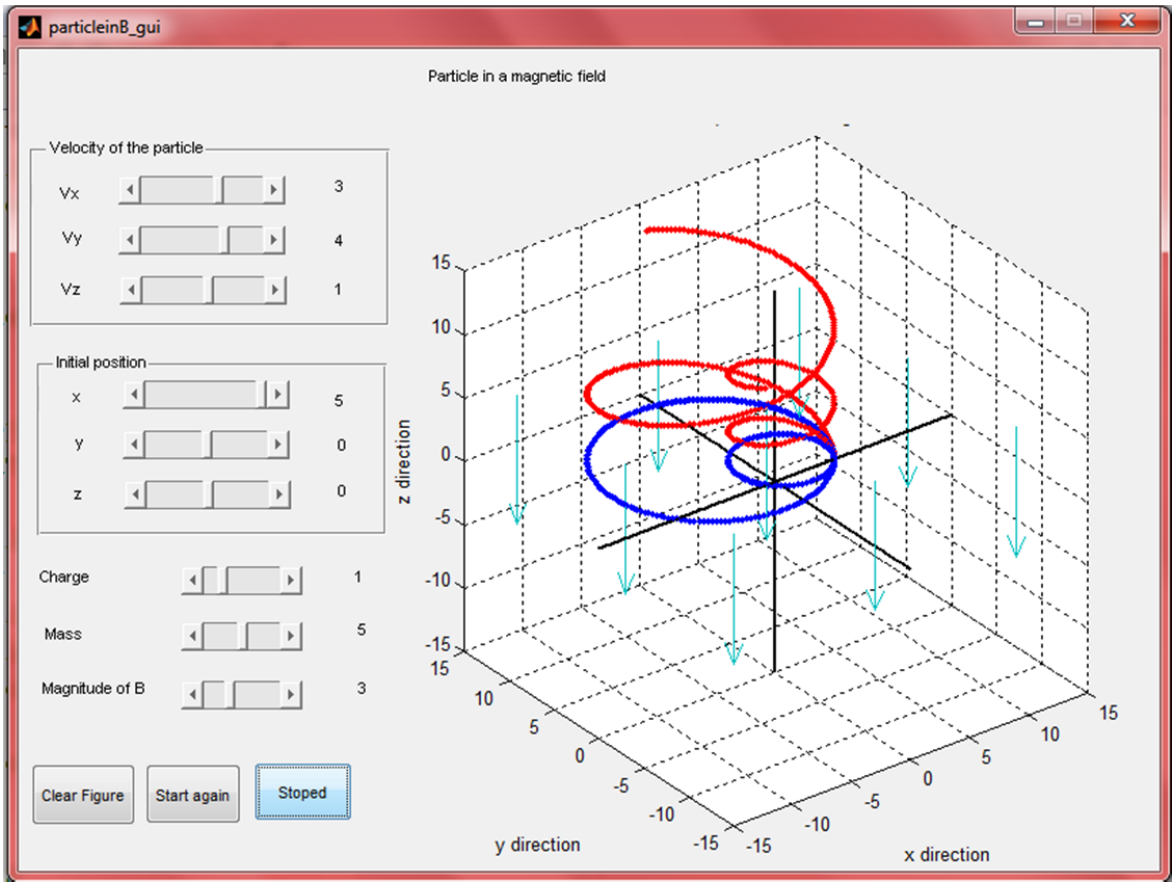
Third Method – GUI approach

The above algorithm is added to a GUI.

Matab Code

```
function varargout = particleinB_gui(varargin)
% PARTICLEINB_GUI M-file for particleinB_gui.fig
%      PARTICLEINB_GUI, by itself, creates a new PARTI-
CLEINB_GUI or raises the existing
%      singleton*.
%
%      H = PARTICLEINB_GUI returns the handle to a new
PARTICLEINB_GUI or the handle to
%      the existing singleton*.
%
%      PARTI-
CLEINB_GUI('CALLBACK',hObject,eventData,handles,...)
calls the local
%      function named CALLBACK in PARTICLEINB_GUI.M
with the given input arguments.
%
%      PARTICLEINB_GUI('Property','Value',...) creates
a new PARTICLEINB_GUI or raises the
%      existing singleton*.  Starting from the left,
property value pairs are
%      applied to the GUI before parti-
cleinB_gui_OpeningFcn gets called.  An
%      unrecognized property name or invalid value
makes property application
%      stop.  All inputs are passed to parti-
cleinB_gui_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose
"GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help
particleinB_gui

% Last Modified by GUIDE v2.5 12-Jun-2011 19:11:28

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
```

Result


particleinB_gui
Particle in a magnetic field
Velocity of the particle
Vx
3
Vy
4
Vz
1
Initial position
x
5
y
0
z
0
Charge
1
Mass
5
Magnitude of B
3
Clear Figure
Start again
Stoped
z direction
y direction
x direction

Crossed electric and magnetic fields

This model simulates the motion of a particle with mass *m* and charge *q*, in electric field E and magnetic field B. These equations are fed into solvers in matlab to find the solution numerically and get the position of the particle as a function of time. There is a variety of motion that results using different values of various variables.

(2)

$$a_x = \frac{d^2x}{dt^2} = (V_y B_z - V_z B_y + E_x)$$

$$a_y = \frac{d^2y}{dt^2} = (B_x V_z - V_x B_z - E_y)$$

$$a_z = \frac{d^2z}{dz^2} = V_x B_y - B_x V_y + E_z)$$

To convert 2nd degree diff eq^n into one degree. use hence.

$$\frac{dx}{dt} = V_x$$

$$\frac{dy}{dt} = V_y$$

$$\frac{dz}{dt} = V_z$$

$$\frac{dV_x}{dt} = V_y B_z - V_z B_y + E_x$$

$$\frac{dV_y}{dt} = B_x V_z - V_x B_z - E_y$$

$$\frac{dV_z}{dt} = V_x B_y - B_x V_y + E_z$$

$x \quad x'$
$y \quad y'$
$z \quad z'$
$V_x \quad V_x'$
$V_y \quad V_y'$
$V_z \quad V_z'$

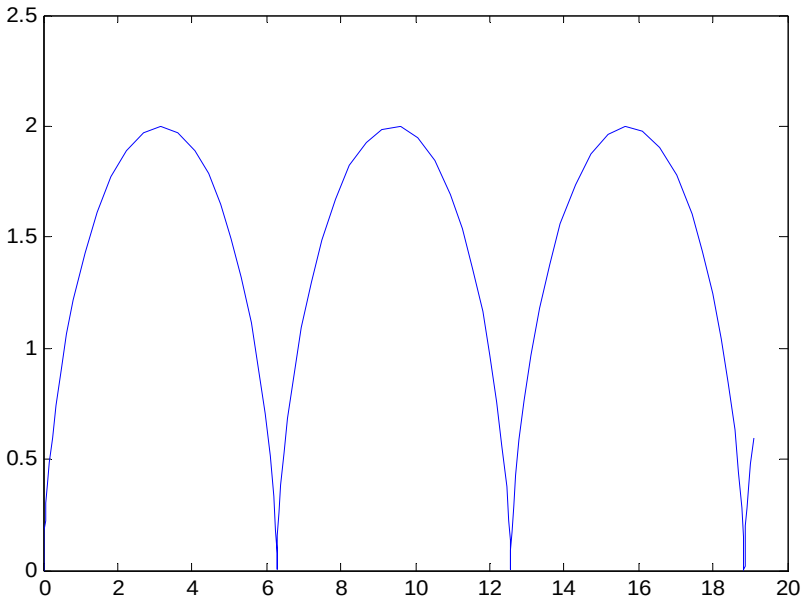
System of ODEs

2d solution

Code

```
% script that simulates a moving particle with some
initial velocity in a
% magnetic field B

v = [0 0 0];   %initial velocity
B = 5;    %magnitude of B
E = 5;
m = 5;          % mass
q = 1;          % charge on particle
r0 = [0 0 0];    % initial position of particle
t = 0;

w = q*B/m;

[t,v] = ode45(@(t,y)e_b(t,y,w,E,B),[0 20],[v(2), r0(2),
r0(3), v(3)]);

plot(v(:,2),v(:,3));
```

The result



Second Method – Using differential equations

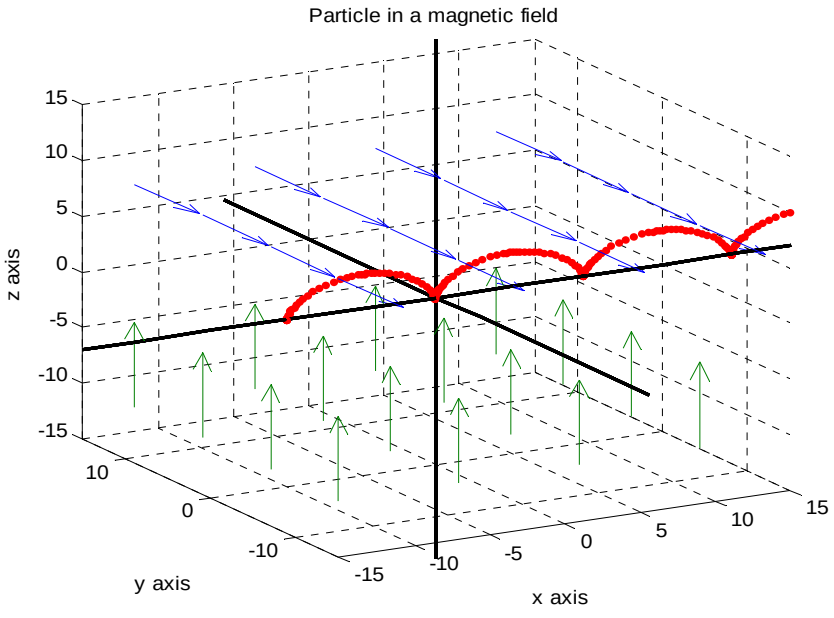
Here in figure below, are the differential equations that I used to find out the trajectory of the particle. These form a set of ODE that can be solved in the matlab built in solver *ode45,* which produces very accurate results.

Code

```
clear all;
close all;
clc;

v0 = [0 0 0]';  %initial velocity
B = [0 -4 0]';  %magnitude of B
E = [0 0 1]';
m = 5;
q = 1;
r0 = [-10 0 0]';
tspan = [0 70];

figure;

%to show B's direction
[x_q,y_q] = meshgrid(-15:8:15,-15:8:15);
z_q=10*ones(size(x_q));

u_q=B(1)*ones(size(x_q));
v_q=B(2)*ones(size(x_q));
w_q=B(3)*ones(size(x_q));
quiver3(x_q,y_q,z_q,u_q,v_q,w_q);
hold all;

% TO show electric field lines
z_q=-10*ones(size(x_q));

u_q=E(1)*ones(size(x_q));
v_q=E(2)*ones(size(x_q));
w_q=E(3)*ones(size(x_q));
quiver3(x_q,y_q,z_q,u_q,v_q,w_q);

%grid on;

%legend('Magnetic field' , 'Electric field');
xlabel ('x axis');
ylabel ('y axis');
zlabel ('z axis');
title ('Particle in a magnetic field')

% draw axis
a = -25:5:25;
```

```
b=zeros(length(a));
plot3(b,b,a,'k','LineWidth',1.5);
hold on;
plot3(a,b,b,'k','LineWidth',1.5);
plot3(b,a,b,'k','LineWidth',1.5);
%rotate3d();

xlim ([-15 15]);
ylim ([-15 15]);
zlim ([-15 15]);

y0 = [r0; v0];
f = @(t,y) [y(4:6); (q/m)*cross(y(4:6),B)+E];
[t,y] = ode23t(f,tspan,y0);

%To show animation
for n=1:length(y)
    plot3(y(n,1),y(n,2),y(n,3),'.r');
    pause(0.00001)
end
```

The result



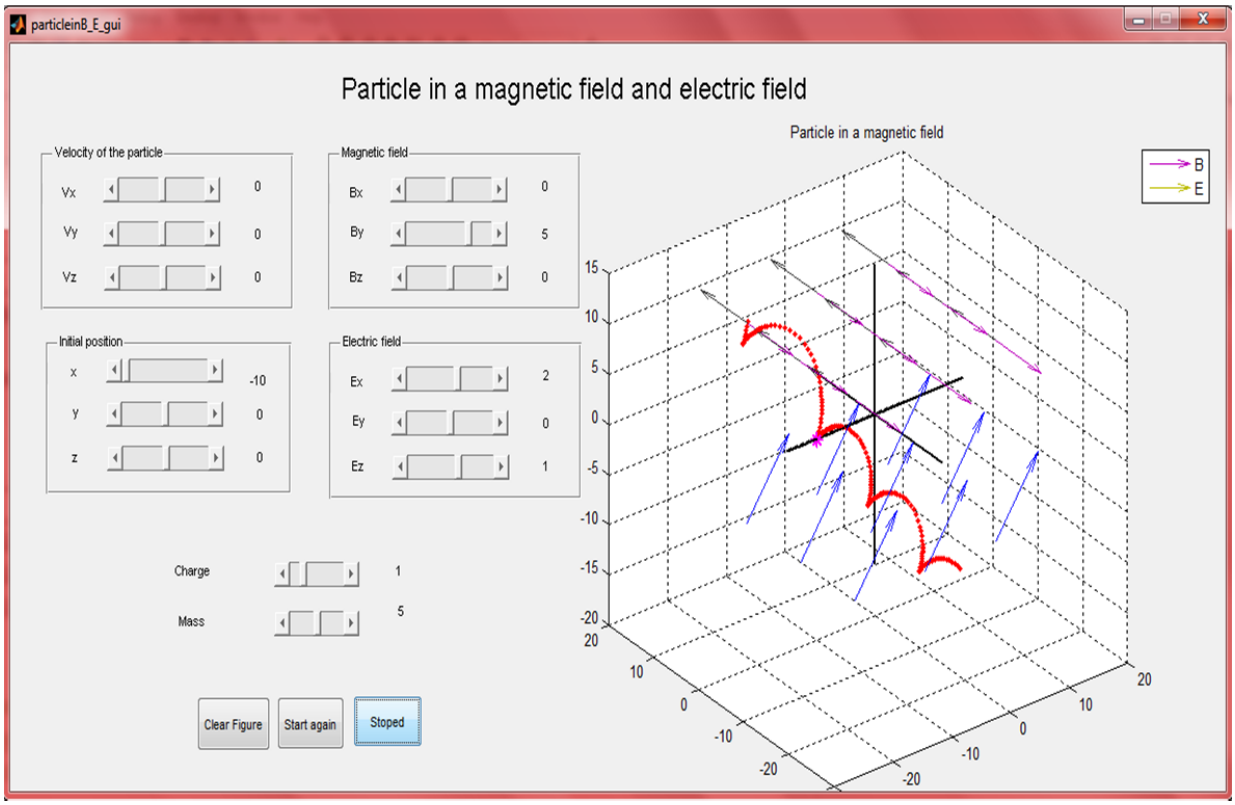
Third Method

To make it interactive, I packed the above code inside a GUI so that a physics teacher can use this to demonstrate the principals of motion of a particle in electro-magnetic field. I developed this for department of physics, IITB. You can change all variable and compare it with motion of particle in previous configuration so that a comparison can be performed.

Code

```
function varargout = particleinB_E_gui(varargin)
% PARTICLEINB_E_GUI M-file for particleinB_E_gui.fig
%      PARTICLEINB_E_GUI, by itself, creates a new
PARTICLEINB_E_GUI or raises the existing
%      singleton*.
%
%      H = PARTICLEINB_E_GUI returns the handle to a
new PARTICLEINB_E_GUI or the handle to
%      the existing singleton*.
%
%
PARTICLEINB_E_GUI('CALLBACK',hObject,eventData,handles,
...) calls the local
%      function named CALLBACK in PARTICLEINB_E_GUI.M
with the given input arguments.
%
%      PARTICLEINB_E_GUI('Property','Value',...)
creates a new PARTICLEINB_E_GUI or raises the
%      existing singleton*.  Starting from the left,
property value pairs are
%      applied to the GUI before
particleinB_E_gui_OpeningFcn gets called.  An
%      unrecognized property name or invalid value
makes property application
%      stop.  All inputs are passed to
particleinB_E_gui_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose
"GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help
particleinB_E_gui

% Last Modified by GUIDE v2.5 17-Jun-2011 05:04:25

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
```

The result


particleinB_E_gui
Particle in a magnetic field and electric field
Velocity of the particle
Vx 0
Vy 0
Vz 0
Magnetic field
Bx 0
By 5
Bz 0
Initial position
x -10
y 0
z 0
Electric field
Ex 2
Ey 0
Ez 1
Charge 1
Mass 5
Clear Figure
Start again
Stoped
Particle in a magnetic field
B
E

Lotka Volterra model

The Lotka-volterra model describes the dynamics of interactions of biological systems in which two species interact, a predator and its prey. A common example is rabbits and foxes. The Lotka–Volterra system of equations is an example of a Kolmogorov model, which is a more general framework that can model the dynamics of ecological systems with predator-prey interactions, competition, disease, and mutualism.

The model is governed by the following system of differential equations

$$R_t = aR - bRF$$
$$F_t = ebRF - cF$$

Where

- R is the population of rabbits,
- F is the population of foxes,
- a is the natural growth rate of rabbits in the absence of predation,
- c is the natural death rate of foxes in the absence of prey,
- b is the death rate of rabbits per interaction with a fox,
- e is the efficiency of turning eaten rabbits into foxes.

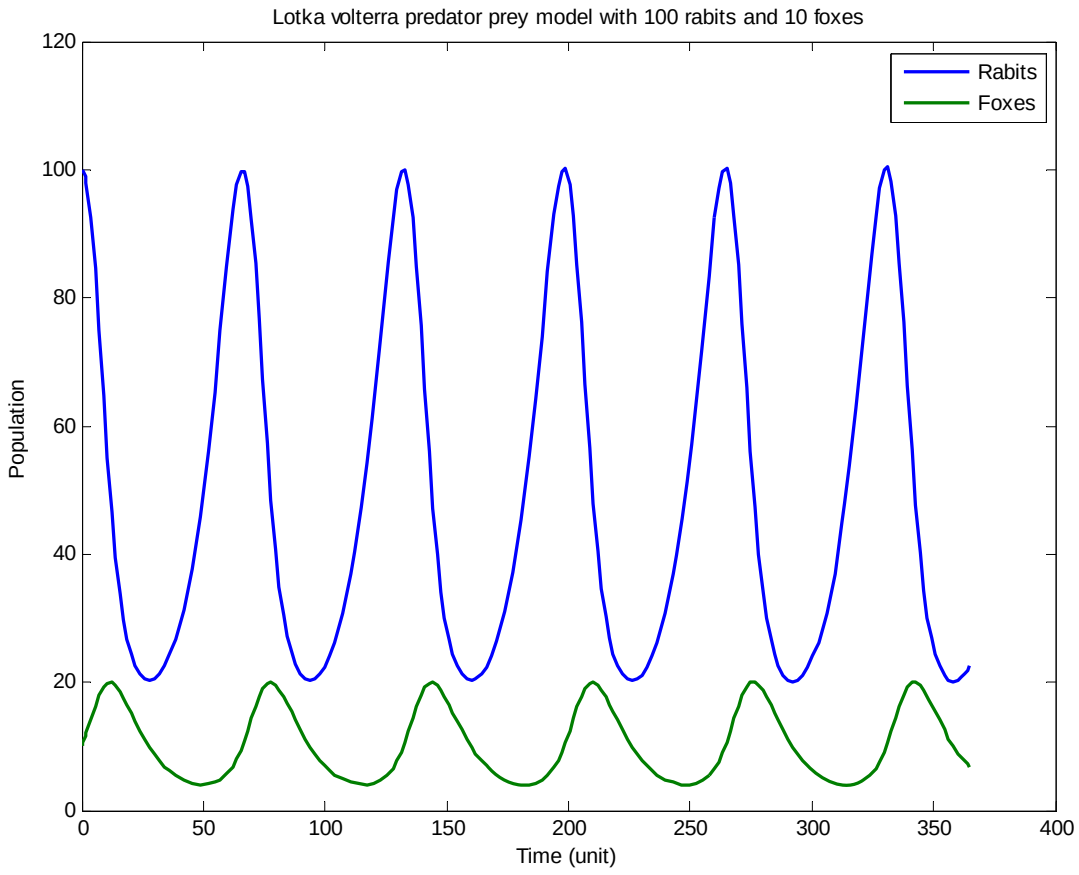
These equations are a pair of first-order, non-linear, differential equations and are coupled so they have to be solved simultaneously in ode45.

Matlab code

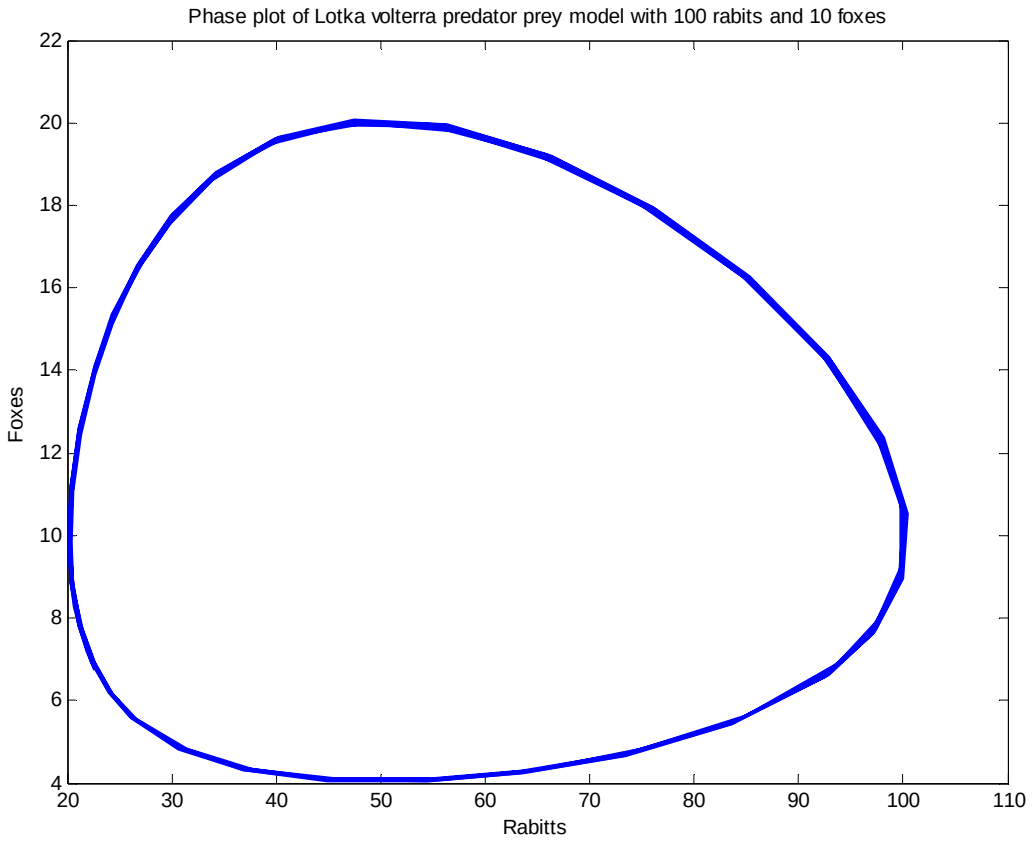
```
close all;
clear all
clc
%Lotka volterra predator prey model
[t,x] = ode45(@lotka_sim, [0, 365], [100, 10])
plot(t,x(:,1),'lineWidth',1.5);
hold all;
plot(t,x(:,2),'lineWidth',1.5);

xlabel('Time (s)');
ylabel('Population');
title('Lotka volterra predator prey model with 100
rabits and 10 foxes');
legend('Rabits','Foxes');
```

```
% To plot the phase plot
figure
plot(x(:,1),x(:,2),'lineWidth',1.5);
xlabel('Rabitts');
ylabel('Foxes');
title('Phase plot of Lotka volterra predator prey model
with 100 rabits and 10 foxes');
```

The result



This curve gives quite an idea of how particular specie of ecosystem varies at different times. This is the predator prey model and shows number of rabbits and foxes at a function of time.



The phase plot of lotka-volterra models represents al the states that the system can assume and what states can follow the given states. Each point on the curve gives a particular number of rabbits (x axis) and foxes (y axis), which is a valid state that the system can assume. The trajectory of the model is periodic unlike the chaotic systems that are discussed in the following sections.

Deterministic chaos

Chaos theory studies the behaviour of dynamical systems that are highly sensitive to initial conditions; an effect which is popularly referred to as the *butterfly effect*. Small differences in initial conditions (such as those due to rounding errors in numerical computation) yield widely diverging outcomes for chaotic systems, rendering long-term prediction impossible in general. This happens even though these systems are *deterministic*, meaning that their future behaviour is fully determined by their initial conditions, with no random elements involved. In other words, the deterministic nature of these systems does not make them predictable. This behaviour is known as *deterministic chaos*, or simply chaos.

A well-known example is weather, in which lies the roots of the chaos theory. Apart from these, chaotic processes are observed in a variety of systems including electrical circuits, lasers, oscillating chemical reactions, fluid dynamics, and mechanical and magneto-mechanical devices, as well as computer models. In numerical analysis, the newton-raphson method can lead to chaotic iterations if the function has no roots.

The systems studied in chaos theory are deterministic. If the initial state were known exactly, then the future state of such a system could be predicted due to chaos. However, in practice, knowledge about the future state is limited by the precision with which the initial state can be measured.

Lorenz attractor

The very first example is a Lorenz attractor which contains the roots of invention of deterministic chaos. Lorenz attractor is an example of a non-linear dynamic system corresponding to the long-term behaviour of the Lorenz oscillator which is a 3-dimensional dynamical system that exhibits chaotic flow, noted for its lemniscate shape. It is chaotic in the sense that the state of the dynamical system(the three variables of the three dimensions) evolves over time in a complex, non-repeating pattern.

This non repeating pattern is characterised by *sensitive dependence* on *initial conditions,* where a small change at one place in a nonlinear system can result in large differences to a later state. This is known as butterfly effect. For example, the presence or absence of a butterfly flapping its wings could lead to creation or absence of a hurricane.

This term was first used by Edward Lorenz in relation to his meteorological work, who studied weather and related differential equations that led to new advances in mathematics, chaos theory. He discovered strange attractor (Lorenz attractor) and coined the term *butterfly effect*.

The reason why whether prediction is so difficult and forecasts are so erratic is no longer thought to be the complexity of the system but the nature of the chaotic DEs modelling them. Small changes (even infinitesimal) in initial conditions can bring widely different results. So, accurate weather prediction depends critically on the accuracy of the measurements of the initial conditions.

The original equations that Edward Lorenz used are far too complex to consider here, we can consider the following much simpler system that has essentially the same chaotic features.

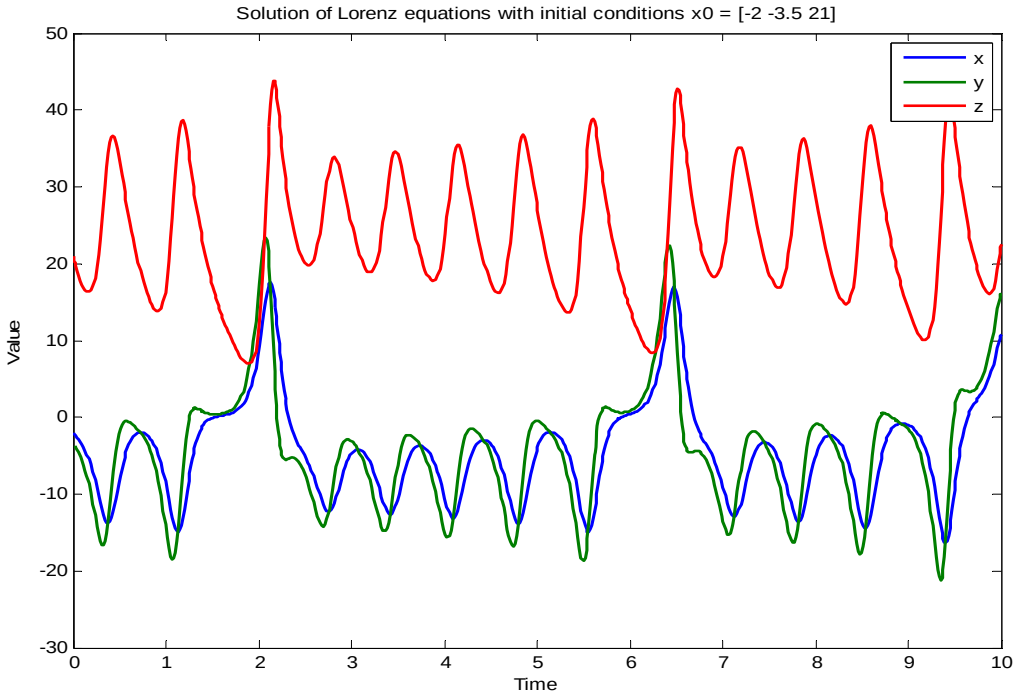
$$dx/dt = 10(y - x),$$
$$dy/dt = -xz + 28x - y,$$
$$dz/dt = xy - 8z/3.$$

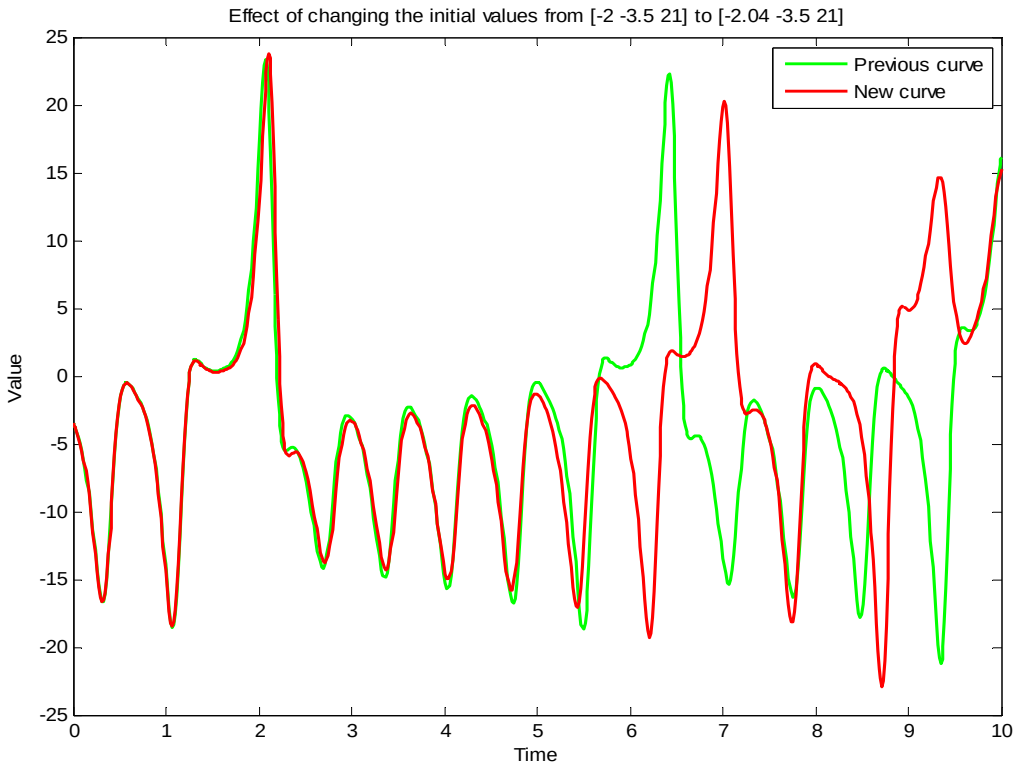
This system of ODE can be solved in matlab ODE solver. The idea to illustrate the effect is to solve the DEs with certain initial conditions, plot the solution, then change the initial conditions very slightly and superimpose the new solution over the old one to see how much it has changed.

Matlab code

```
function f = lorenz_sim(t, x)
f = zeros(3,1);
f(1) = 10 * (x(2) - x(1));
f(2) = -x(1) * x(3) + 28 * x(1) - x(2);
f(3) = x(1) * x(2) - 8 * x(3) / 3;


% To plot the solution of lorenz attractor
x0 = [-2 -3.5 21]; % initial values in a vector
[t, x] = ode45(@lorenz_sim, [0 10], x0);
figure;
plot(t,x,'LineWidth',1.5) % plots 3 plot of x,y and z
title('Solution of Lorenz equations with initial condi-
tions x0 = [-2 -3.5 21]')
legend('x','y','z');
xlabel('Time');
ylabel('Value');

% To see effect of changing the initial values
figure;
plot(t,x(:,2),'g','LineWidth',1.5);
hold on;
x0 = [-2.04 -3.5 21];
[t, x] = ode45(@lorenz_sim, [0 10], x0);
plot(t,x(:,2),'r','LineWidth',1.5)
title('Effect of changing the initial values from [-2 -
3.5 21] to [-2.04 -3.5 21]');
xlabel('Time');
ylabel('Value');
legend ('Previous curve', 'New curve');
```

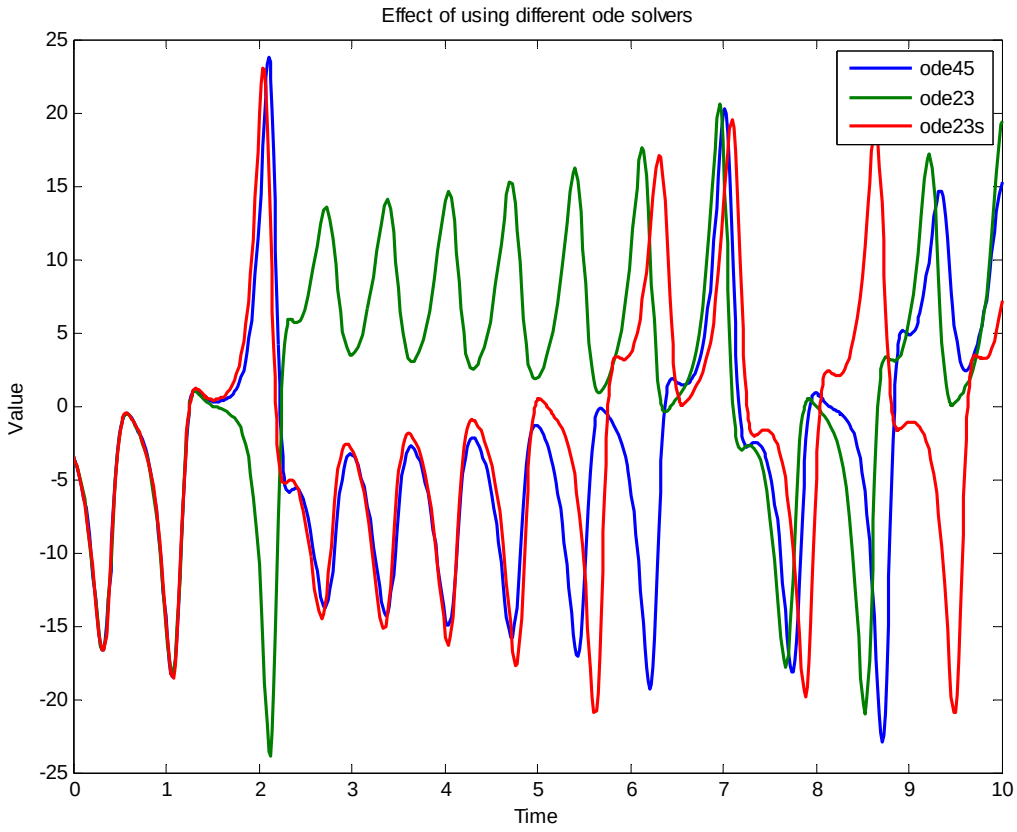
The result


Solution of Lorenz equations with initial conditions x0 = [-2 -3.5 21]
x
y
z
Value
Time


Effect of changing the initial values from [-2 -3.5 21] to [-2.04 -3.5 21]
Previous curve
New curve
Value
Time

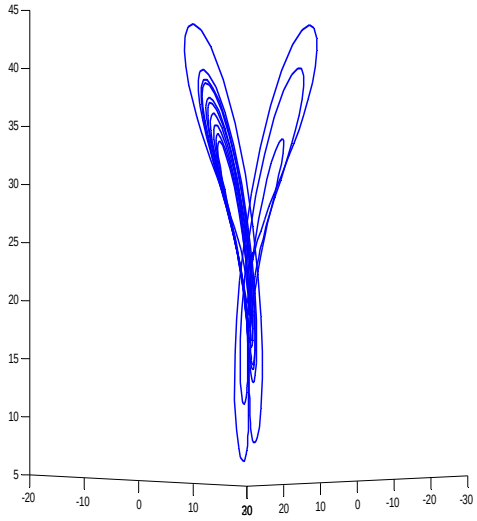
This is how the chaotic solution [x,y,z] of Lorenz equations looks like and the effect of changing initial conditions even by a small amount. In the second curve only **y(t)** is plotted vs **t** for initial conditions and final conditions, to clearly illustrate the effect. With only 2% change in initial conditions from -2 to -2.04, there's a large discrepancy in the solution of y(t). The two graphs are practically indistinguishable until t is about 1.5. The discrepancy grows quite gradually, until t reaches about 6, when the solutions suddenly and shockingly flip over in opposite directions. As t increases further, the new solution bears no resemblance to the old one.

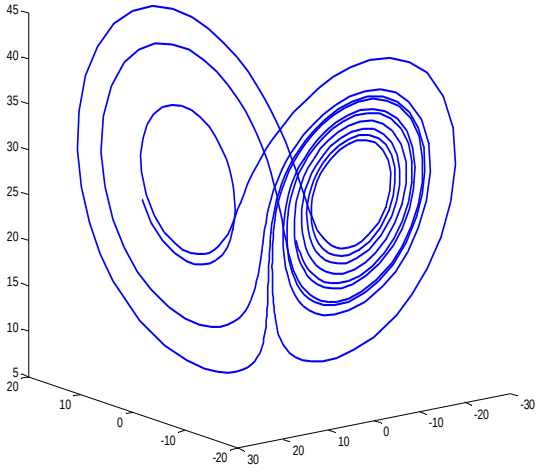
Now let's see the effect of solving the equations using a different ODE solver in matlab. There are different ODE solvers like ode45, ode23 and ode23s all have different numerical accuracies. For that change the code a little, use ode23 and ode23s instead of ode45 and obtain the graph below.



Solution of ode45 and ode23 deviate from t>1.5 while solution of ode45 and ode23s vary significantly from t>5. The explanation is that ode23, ode23s and ode45 all have numerical inaccuracies (if one could compare them with the exact solution—which incidentally can't be found). However, the numerical inaccuracies are different in the three cases. This difference has the same effect as starting the numerical solution with very slightly different initial values.

Now let's have a final look at the famous butterfly picture of chaos from which the name, butterfly effect originated. The picture is simply the plot of x against z against y as time increases and it's called a *phase plane plot*.





This is the same 3d plot, rotated in axis to show it from different angles. This is **phase plane plot** that represents various states that system can assume and looks like a butterfly. Unlike in lotka-volterra phase plot, it's not periodic but rather complex and chaotic. You can also create an animation that shows how a Lorenz system evolves over time

Mandelbrot set

The term Mandelbrot set is used to refer both to a general class of fractal sets and to a particular instance of such a set. In general, a Mandelbrot set marks the set of points in the complex plane such that the corresponding Julia set is connected and not computable.

"The" Mandelbrot set is the set obtained from the quadratic recurrence equation

$$z_{n+1} = z_n^2 + C$$

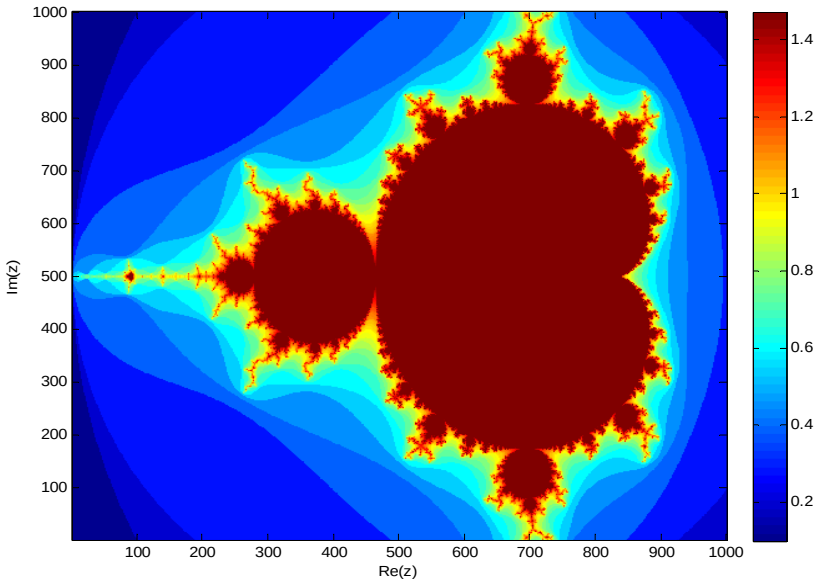
with $z_0 = C$,, where points C in the complex plane for which the orbit of z_n does not tend to infinity are in the set. Setting z_0 equal to any point in the set that is not a periodic point gives the same result.

The Mandelbrot sets are based on the idea of choosing two complex numbers z_0 and c, and then repeatedly evaluating $z_n=z_{n-1}^2+c$. If z_n does not tend to infinity the point (z_0,c) is in the set (colored blue or black in the pictures on this page). If we consider all possible z_0 and c then we end up with a four dimensional set, which is rather hard to visualise.

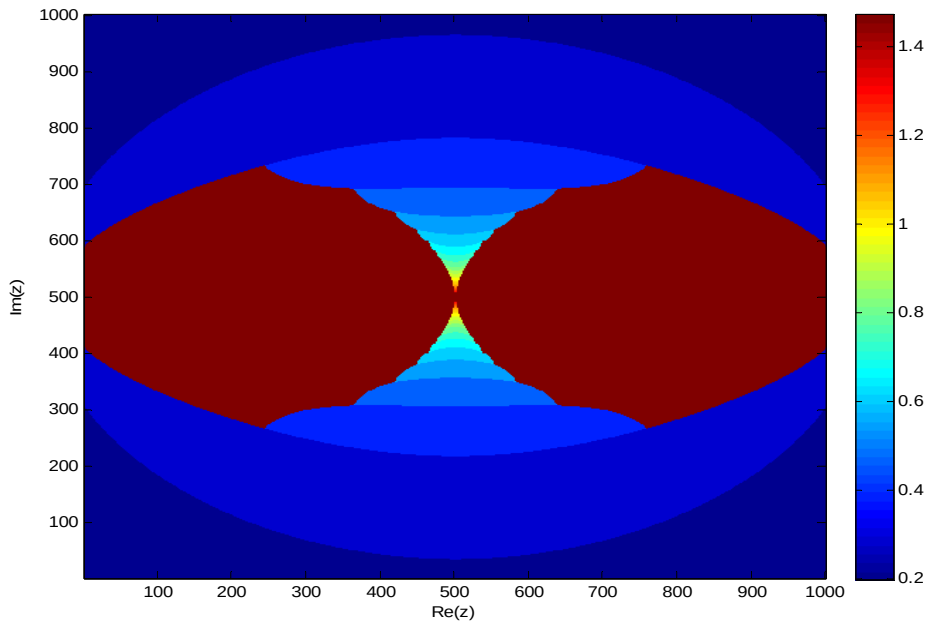
Matlab Code

```
function M=mandelbrot(zMax,N)
MAT=1000;
zmax_l=linspace(-zMax,zMax,MAT);
zmax_l_x=linspace(-2*zMax,0.7*zMax,MAT);
[MATx,MATy]=meshgrid(zmax_l_x,zmax_l);
C=zeros(MAT,MAT);
M=zeros(MAT,MAT);
Z=MATx+i*MATy;

for k=1:MAT
    for j=1:MAT
        M(k,j)=escapeVelocity(0,Z(k,j),N);
    end
end

imagesc(atan(0.1*M));
axis xy;
xlabel('Re(z)'); ylabel('Im(z)');
colorbar;
```

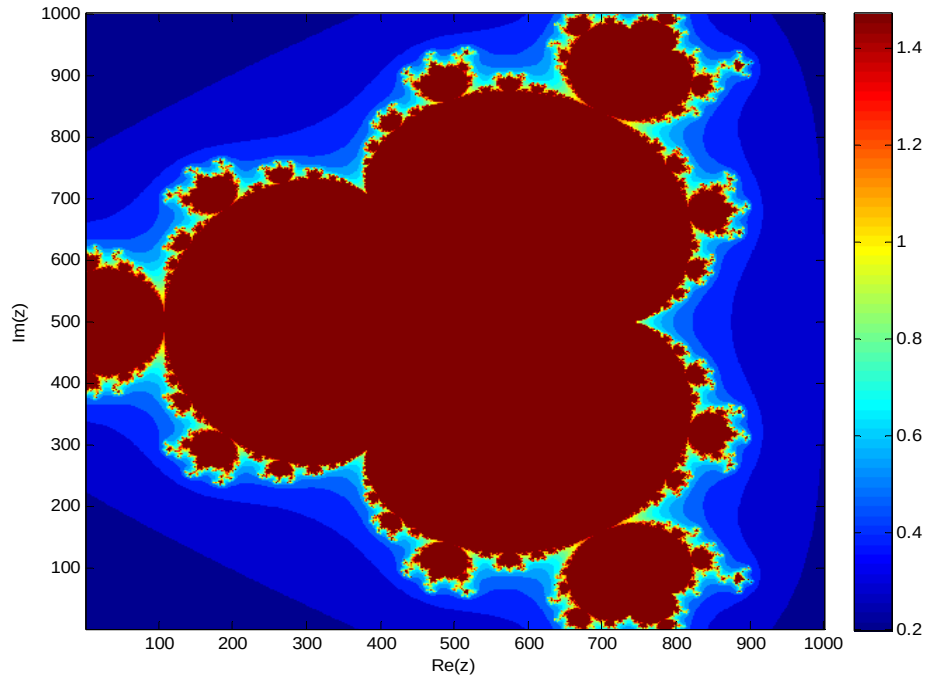
The result



mandelbrot(1,100) $z = x^2 + c$



Mandelbrot set using **z=sin(z) + c**



Mandelbrot set using $\mathbf{z = z^4 + c}$

Julia sets

Suppose $\mathbf{p(z)=z^2+c}$ is a complex quadratic polynomial (the value c is a given complex number). A Julia set is created by *iterating* p(z). That is, the output of p(z) is fed back in as input for a new value of z. Therefore, one can rewrite the polynomial as

$$z_{n+1}=z_n^2+c.$$

For any initial value, z_0, iterations of p(z) will either be bounded or unbounded. Suppose the set of all z-values that result in bounded iterations of p(z) is denoted by B. Let the set of z-values that result in unbounded iterations of p(z) be denoted D. It turns out that the sets B and D intersect. The intersection of these two sets is called a Julia set.

This is the more technical Wikipedia definition, which is a bit more mathematical. "In the context of complex dynamics, a topic of mathematics, the Julia set and the Fatou set are two complementary sets defined from a function. Informally, the Fatou set of the function consists of values with the property **that all nearby values behave similarly** under repeated iteration of the function, and the Julia set consists of values such **that an arbitrarily small perturbation** can cause drastic changes in the sequence of iterated function values. Thus the behaviour of the function on the Fatou set is 'regular', while on the Julia set its behaviour is 'chaotic'. "

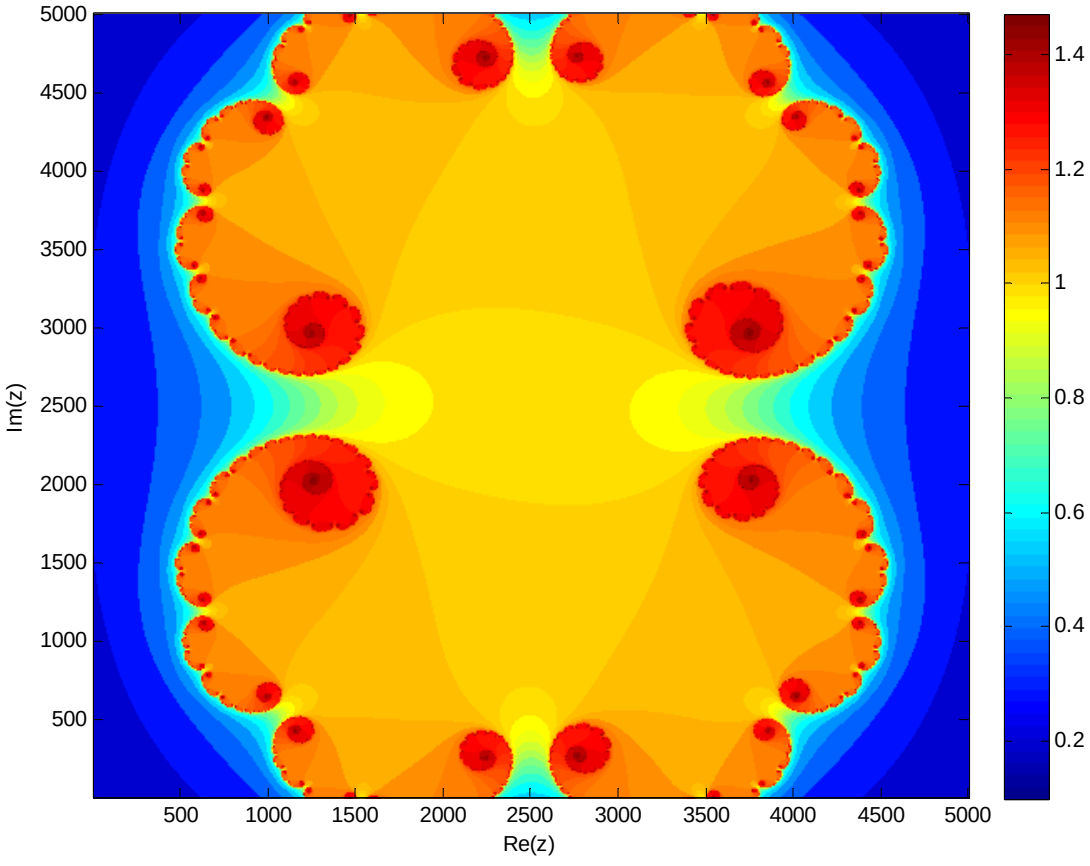
There are a variety of algorithms to color the escaping points in the complex plane and according to that there are a variety of color schemes. I have used the popular escape velocity algorithm to draw these beautiful fractels of Julia sets in matlab.

Matlab code

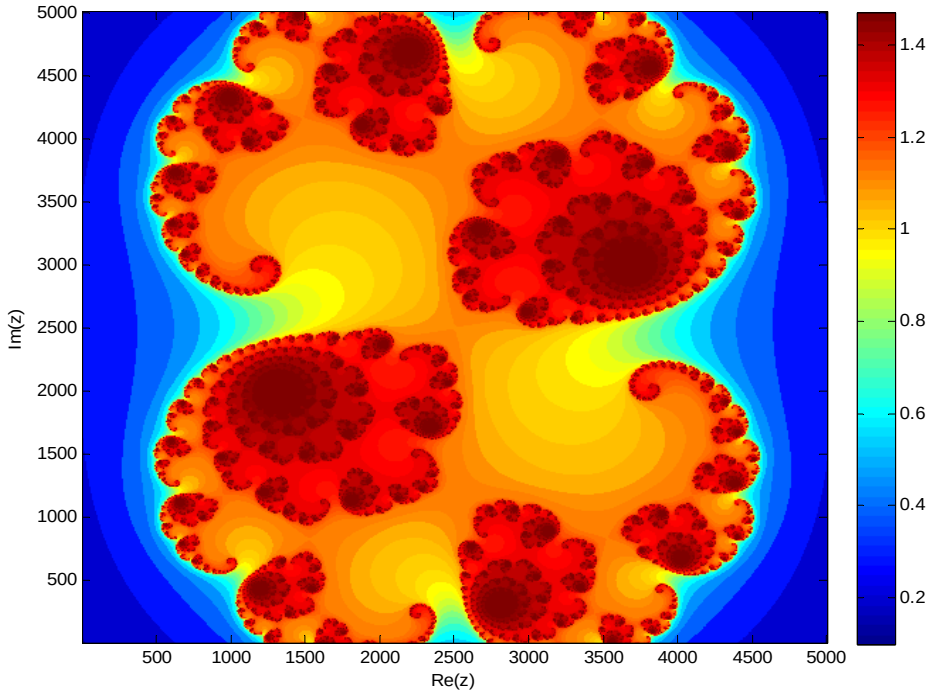
```
function n=escapeVelocity(Z0,c,N)
n=0;
Z=Z0;

while abs(Z)<=2 && n<N
    Z=((Z^2)+c);
    n=n+1;
end

function M=julia(zMax,c,N)
MAT=5000;
mat=linspace(-zMax,zMax,MAT);
[MATx,MATy]=meshgrid(mat,mat);
Z=zeros(MAT,MAT);
M=zeros(MAT,MAT);
Z=MATx+i*MATy;
```

```
for k=1:MAT
    for j=1:MAT
        M(k,j)=escapeVelocity(Z(k,j),c,N);
    end
end

imagesc(atan(0.1*M));
axis xy;
xlabel('Re(z)'); ylabel('Im(z)');
colorbar;
```

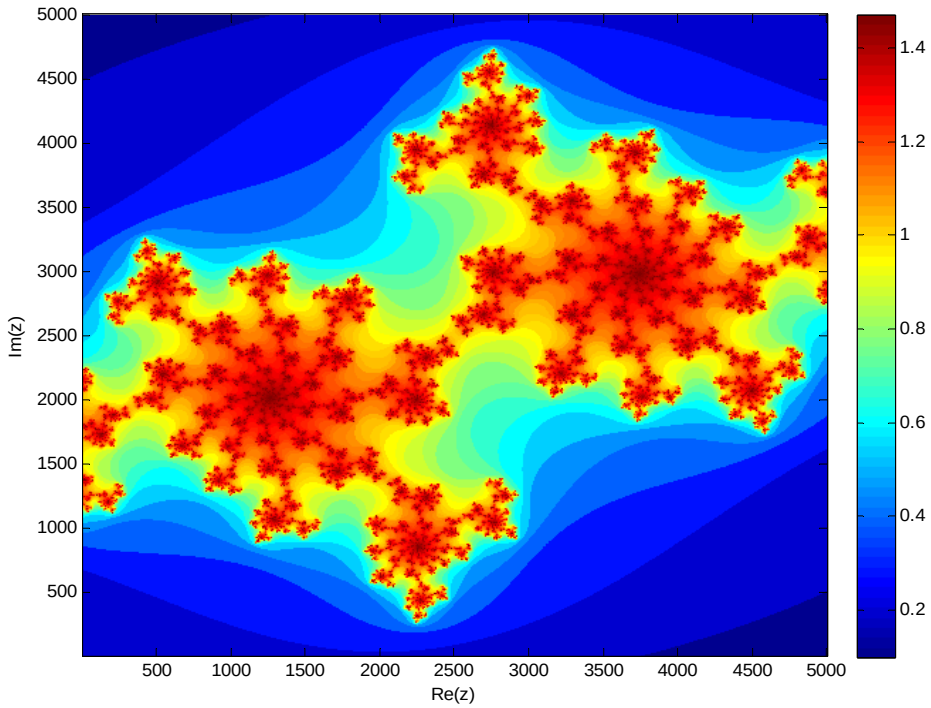
The results



M=julia(1,0.285+0.001i,100)



M=julia(1,0.285+0.01i,100)



M=julia(1,-0.70176-0.3842i,100)

Studying the chaotic nature of julia set

I tried to use Julia set to create a pseudorandom number generator. A pseudorandom number generator is a deterministic algorithm, although its evolution is deliberately made hard to predict; so it's used for a lot of practical purposes while a hardware random number generator, may be non-deterministic. The results of this experiment is as below.

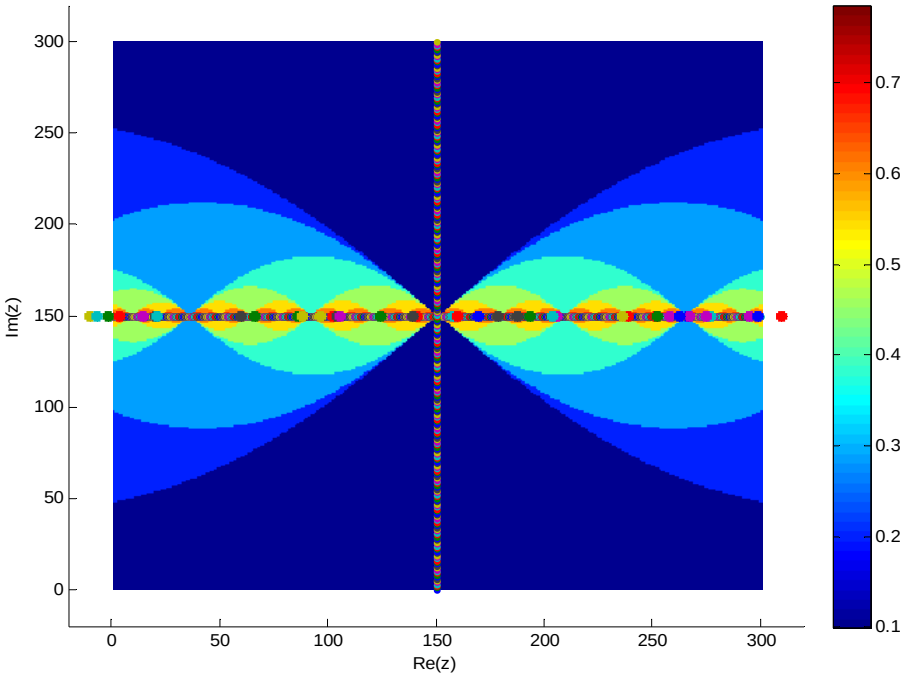
Matlab Code

```
clc
close all;
clear all;


c=-2;
freq = 100;
freq2 = 100;
Z0 = 1.2 + 0i;

clc;
Z=Z0

W = linspace(0,0,freq);

for n= 1:freq
    W(n)=Z;
    Z = Z^2 + c;
end

% data with mean
figure
title ('Random sample taken from Julia set iterations
bw [-2 2]. Taken for sample size=100')

ylabel ('Data');
hold on;
plot(W);
ylim([-6, 6])

%plot(0:0.1:freq,2,'--')
%plot(0:0.1:freq,-2,'--')

plot(0:0.1:freq,mean(W),'m')
legend('Data','Mean')
```

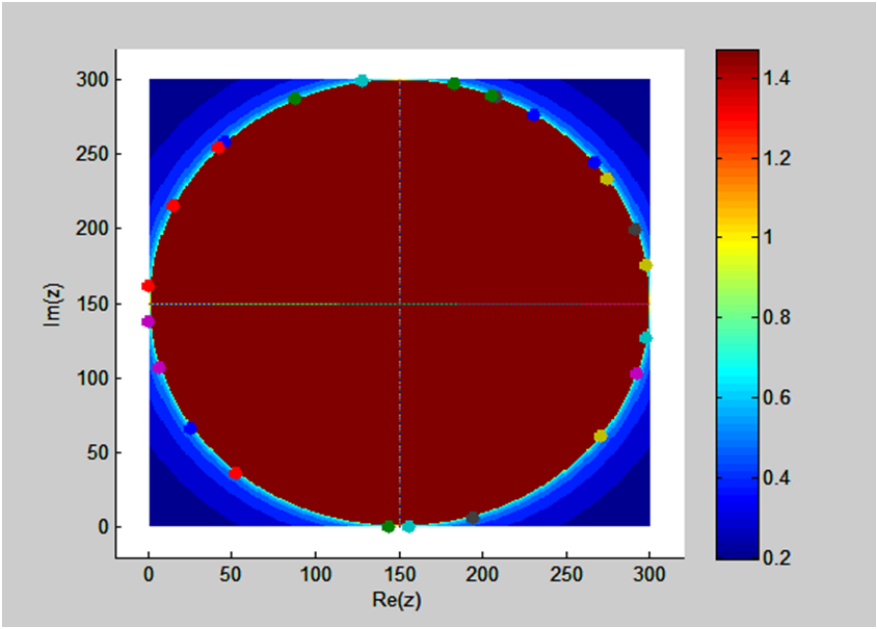
```
% TO STUDY MEAN WITH SAMPLE SIZE
figure
hold on;

Z0 = 1.2 + 0i;

clc;

mn = linspace(0,0,freq2-10);
sum = zeros(freq2-10,1);

for k = 11:freq2
    Z=Z0;
    for n=1:k
 %        W_2(k,n)=Z;
         sum(k-10) = sum(k-10) + Z;
         Z = Z^2 + c;
    end
    mn(k-10) = sum(k-10)/k;
end

plot(11:1:freq2,mn);
hold on;

plot(11:0.1:freq2,0);
title('Studying relation of mean with sample size');
xlabel('Sample size');
ylabel('Mean(s)');
```

The results

Random nature



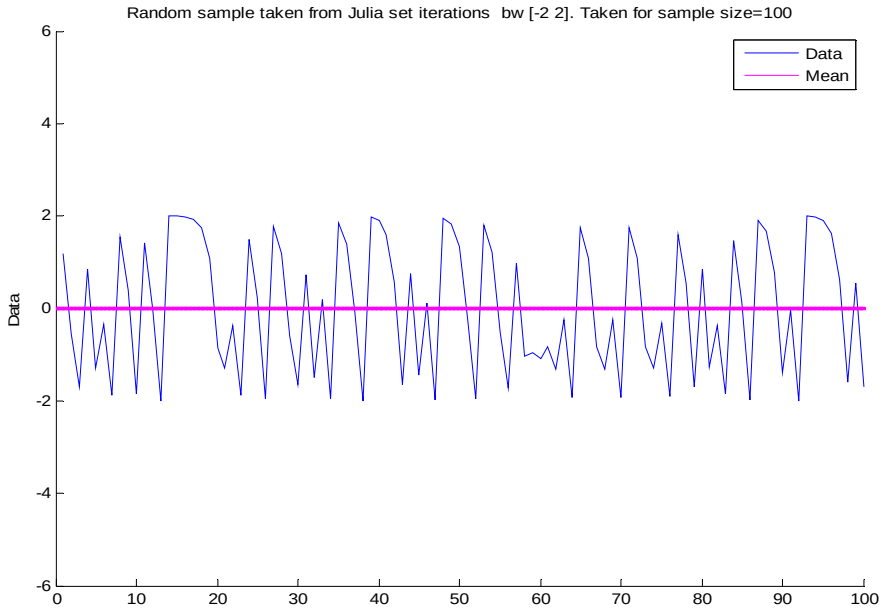
C=-2 and, **x € (-2, 2)**



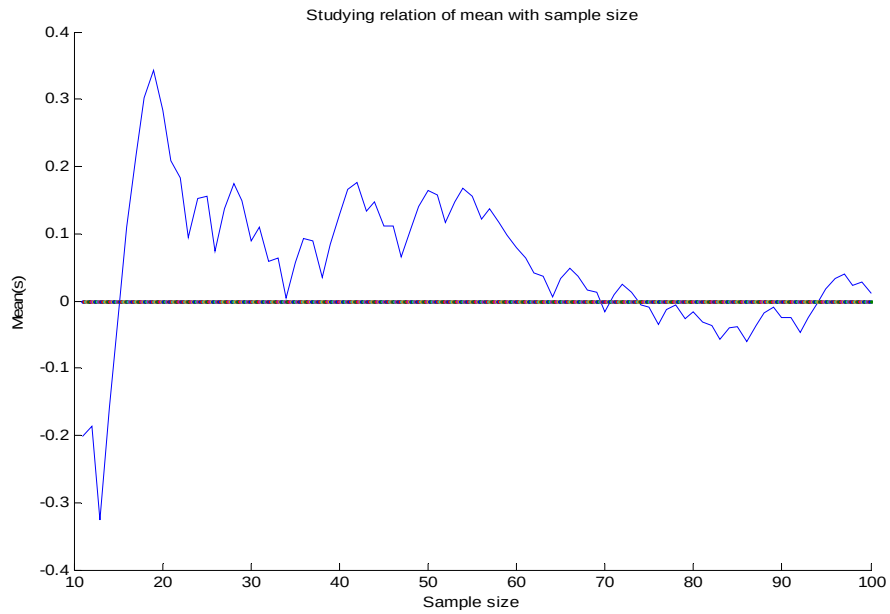
Z0 = cos(ø) + i sin(ø); c = 0;

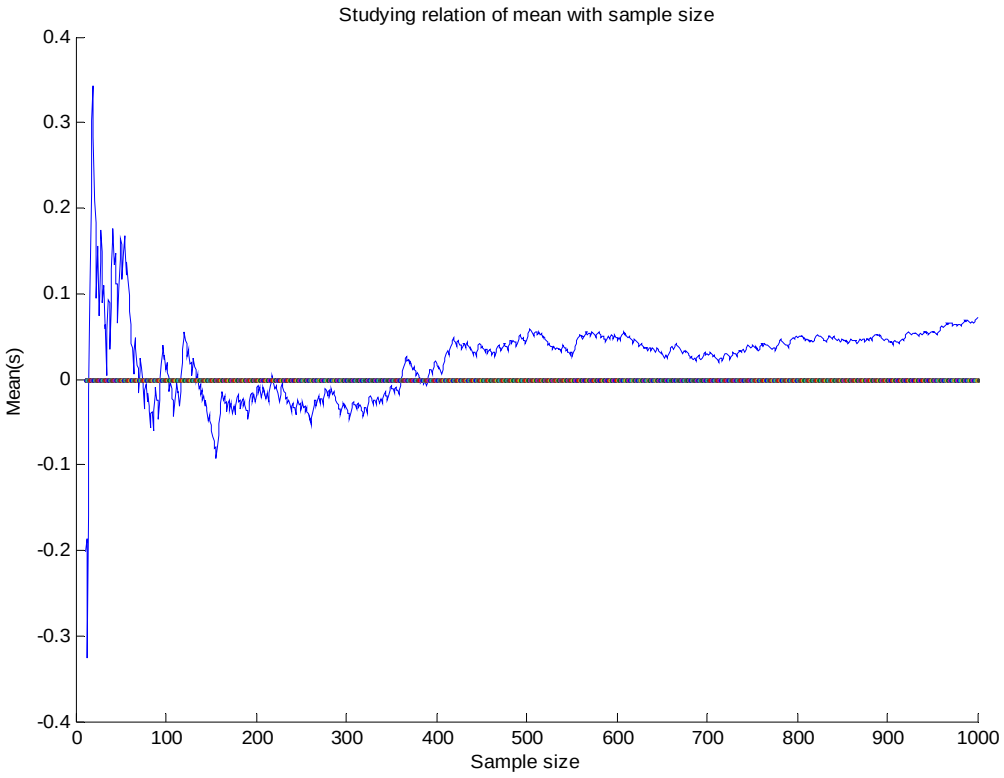
Testing the quality of random numbers generated

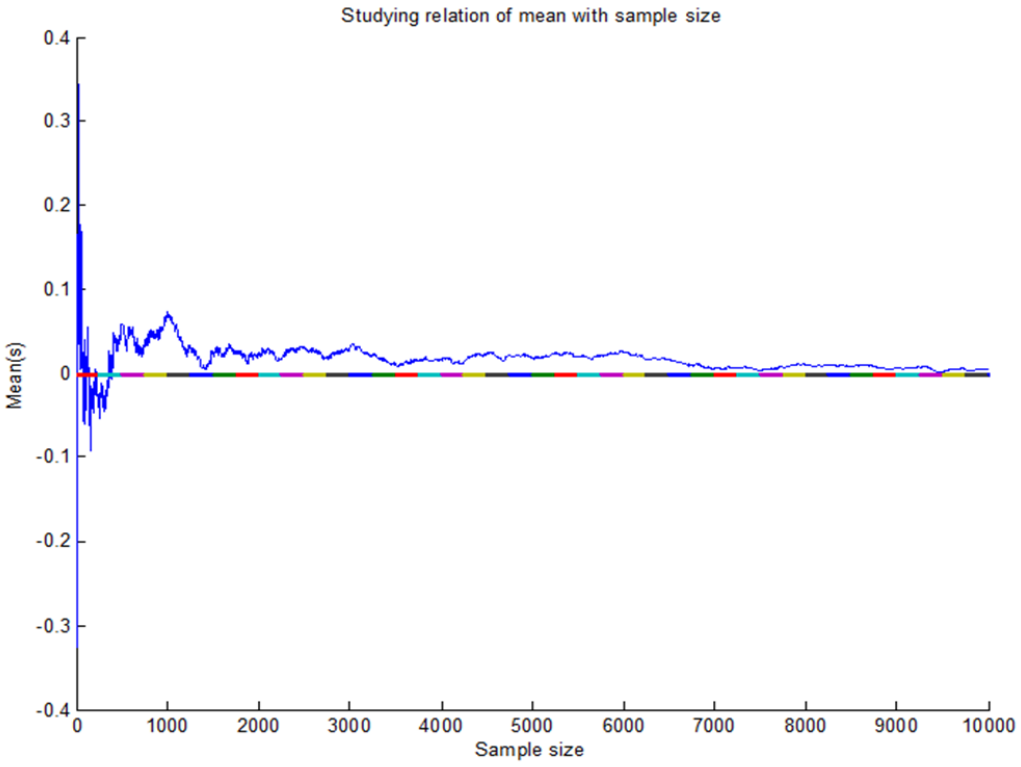
Every time we run the simulation with same parameters, the data I got was exactly the same. This proves that our random number generator is not essentially an unbiased random number generator but in some essence it is deterministic which only appears to be random. This is essentially due to deterministic chaos. These types of random number generators are known as pseudo random number generators.



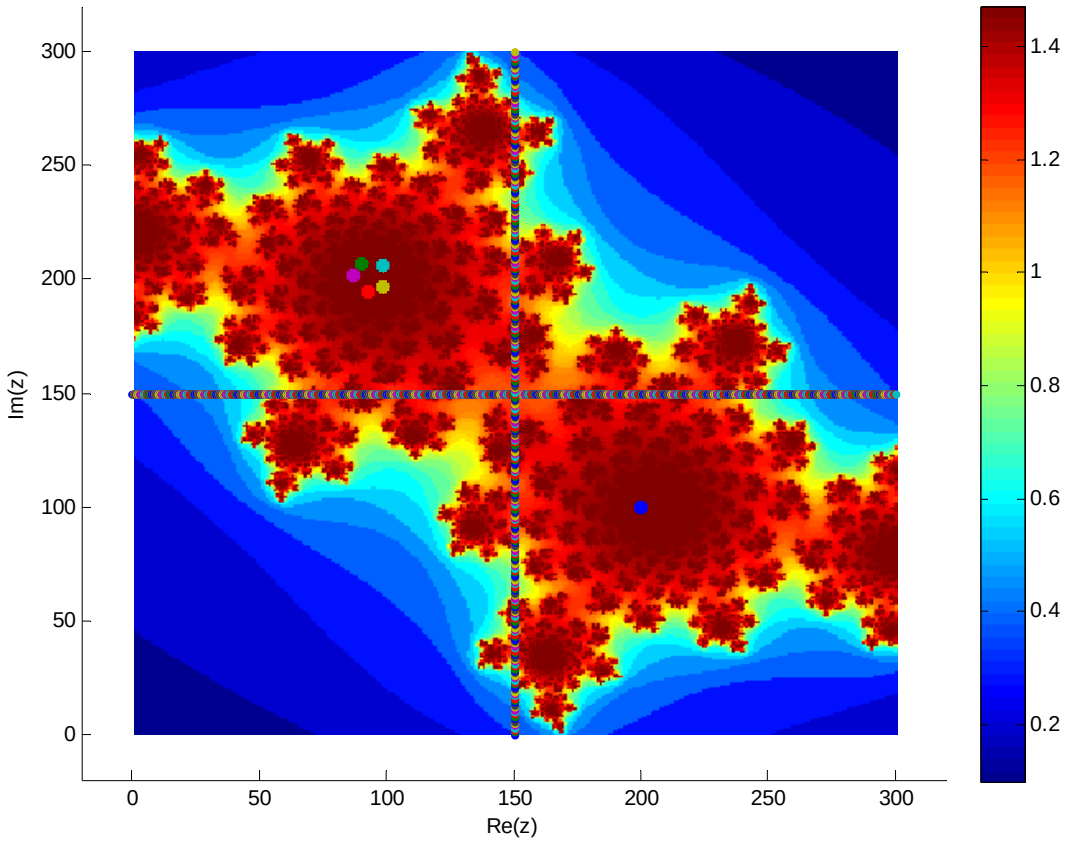
This illustrates deterministic chaos of Julia sets iterations. Sample size = 100




Studying relation of mean with sample size
Mean(s)
Sample size
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
0
100
200
300
400
500
600
700
800
900
1000


Studying relation of mean with sample size
Mean(s)
Sample size
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000

Studying Julia trajectories



To view the chaotic nature of julia set, I edited the matlab codet to plot the trajectory or orbit of a single random point in the complex plane. To make it more illustrative, I plotted the trajectory on the Julia set. The result can be shown in the figure above, the point started with the blue dot and moved chaotically to the other side of origin and rotated there, moving inside. The dense red area can be identified as *points of attraction* of Julia set.

Stochastic systems

Stochastic processes

Stochastic systems, involving stochastic processes or random processes are counterpart of deterministic systems involving deterministic processes. In case of deterministic process we deal with only one possible reality of how the process might evolve under time (as in the case, for example the solution of ODEs), where in a stochastic or random processes there is some indeterminacy in its future evolution described by *probability distributions*. This means that even if the initial conditions (or starting point) is known, there are many possibilities the process might go to, but some paths may be more probable that others.

A stochastic process often involves a random variable (which is a non-deterministic, single quantity having certain probability distributions) whose probability distribution changes as the process evolves, producing a sequence of random variables. In case of discrete time (a very simple case) a stochastic process amounts to a sequence of random variables known as a *time series* as in *Markov chain*. Another basic type of a stochastic process is a *random field*, whose domain is a region of space, in other words, a random function whose arguments are drawn from a range of continuously changing values.

Stochastic processes can also be viewed as functions of one or several *deterministic arguments* (inputs, in most cases regarded as time) whose output values are *random variables*. Random variables corresponding to various times (or points, in the case of random fields) may be completely different.

Stochastic model

A statistical model or a stochastic model (used interchangeably) is used to model a stochastic process. A statistical model is a formalization of relationships between variables in the form of mathematical equations, which describes how one or more random variables are related to one another. The model is statistical as the variables are not deterministically but stochastically related. In mathematical terms, a statistical model is frequently thought of as a pair (Y,P) where Y is the set of possible observations and P the set of possible probability distributions on Y. It is assumed that there is a distinct element of P which generates the observed data. *Statistical inference* enables us to make statements about which element(s) of this set are likely to be the true one.

One major use of these stochastic models is in statistical inference, which is the process of drawing conclusions from data that are subject to random variation, for example, observational errors or sampling variation. Similar terms such as statistical inference, statistical induction and inferential statistics are used interchangeably. Inference is a vital element of scientific advance, since it provides a prediction (based in data) for where a theory logically leads. Initial requirements of such a system of procedures for inference and induction are that the system should produce reasonable answers when applied to well-defined situations and that it should be general enough to be applied across a range of situations.

Inferential statistics uses patterns in the sample data to draw inferences about the population represented, accounting for randomness. These inferences may take the form of: answering yes/no questions about the data (hypothesis testing), estimating numerical characteristics of the data (estimation), describing associations within the data (correlation) and modelling relationships within the data (for example, using regression analysis). Inference can extend to forecasting, prediction and estimation of unobserved values either in or associated with the population

being studied; it can include extrapolation and interpolation of time series or spatial data, and can also include data mining. One primary requirement of inferential statistics is that a randomly chosen sample should tend to exhibit the same properties as the population from which it is drawn.

Familiar examples of processes modelled as stochastic time series include stock market and exchange rate fluctuations, signals such as speech, audio and video, medical data such as a patient's EKG, EEG, blood pressure or temperature, and random movement such as Brownian motion or random walks. Examples of random fields include static images, random terrain (landscapes), or composition variations of a heterogeneous material.

Computer Simulations of random processes and computational statistics

At a most basic level, statistics is concerned with the transformation of raw data into knowledge, which is inferential statistics. There are traditional approaches for making these inferences out of data collected which includes regression, hypothesis testing, parameter estimation, confidence intervals, etc. But with the advent of computers, in present world data sets that analysts must deal with tend to be very large and high-dimensional, so in these situations classical methods in statistics are inadequate and computational statistics techniques are used which involves some techniques that have a strong focus on the exploitation of computing in the creation of new statistical methodology. In computational statistics, computer simulations are used for obtaining the answer.

A computer simulation in this context is a *computer experiment* which mirrors some aspect of real world that appears to be based on random processes or is too complicated to understand properly. In general we build a model that pretends to be real world situation and simulates what goes on. The essence of simulation programs in computer is that the programmer is unable to predict beforehand exactly what the outcome of the program will be. The output of these computer simulation that tend to mimic an actual random process, enables us to make inferences about the future of the process evolving out, depending on some data taken from a sample, that is expected to represent the population under study. There are many techniques, how this is done and some are discussed below with examples.

Random experiment in computers

Suppose if there is an experimenter who want carry out a random experiment of tossing a fair coin, say 100 times to study the result of this experiment. A person who don't know what computer simulations are has no choice but to flip the *fair* coin 100 times and noting the output. It is important to recall what exactly a fair coin is. A fair coin is a theoretical thing whose H/T ratio will approach 0 as the no of flips approach infinity. So in order to verify whether the coin is fair or not, he has to flip the coin infinite times or large no of times in practical, only he can guarantee that the result of his 100 coin flip experiment is not biased. This seems to be very tedious work for the experimenter.

Suppose anyhow he already got a fair coin, still he has a lot of work to flip the coin 100 times. Of course flipping the coin a 100 times won't give him equal number of heads and tails even for a fair coin. Now suppose he wants to see what happens if he flips again 100 times? Again the result will not be equal; in fact it will not match the previous result. There may be many question in the experimenter's mind, he may want to flip the coin 1000 times and see what happens. This seems very tedious using a fair coin in actual, instead we have better and easy way of carrying out this experiment in computer. Here I will show you how to conduct the experiment in computer

Experiment: Flipping the coin 100 times

In matlab we have random number generators for this job, which precisely in nature is a pseudo random number generator like the one I programmed using Julia set. Although it can generate fairly good random numbers, which appear really random, but it is pseudo in the sense that a computer is a deterministic machine and the sequence generated is completely determined by a relatively small set of initial values, called the PRNG's state. In theory, a pseudo random number generator only approximates the properties of a random number and the sequence is not truly random but they are practically unpredictable and this will solve our purpose of a random number generator.

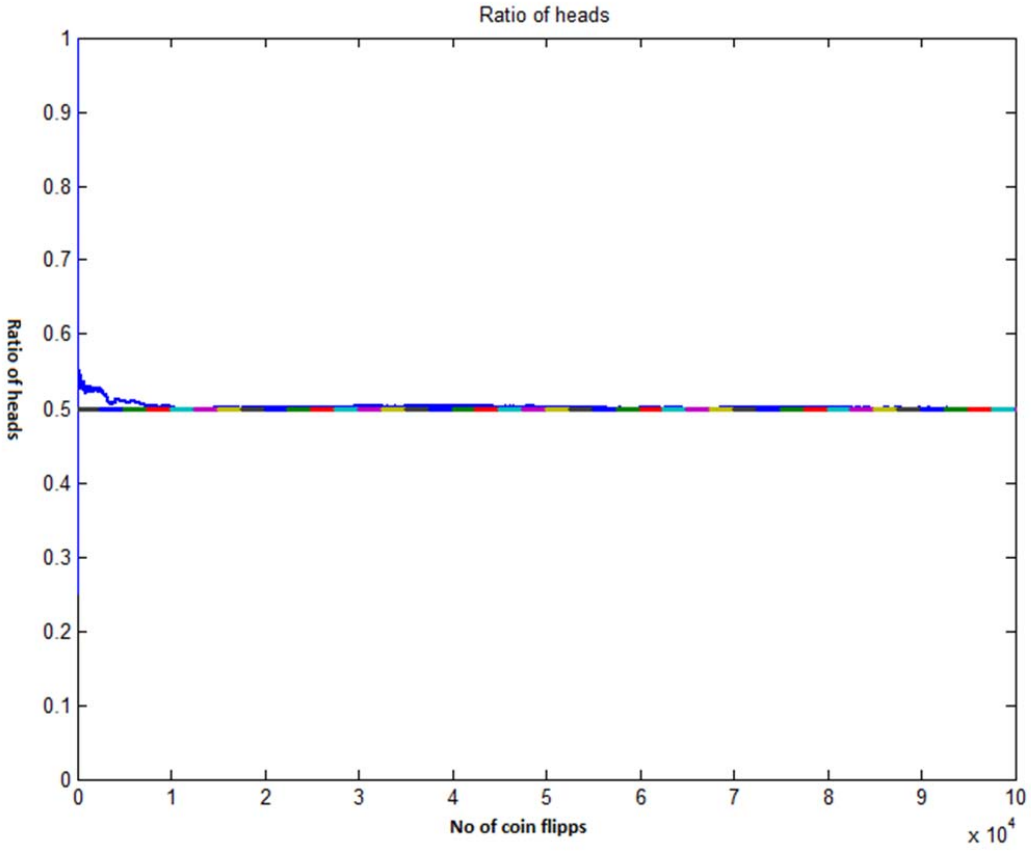
Now the idea is that we can exploit this random number generator which we are assuming is sufficiently random (like the experimenter assumed that the coin is really fair), to simulate a coin flip. In matlab the function for this purpose is *rand*, which generates a uniformly distributed random number between 0 and 1 which is sufficiently random and has been used successfully by many in many of these types of experiments. We can easily mimic a coin flip by using this function such that if the number it generates is less than 0.5, it's a head otherwise its tail.

Although careful mathematical analysis is required to have any confidence a PRNG generates numbers that are sufficiently "random" to suit the intended use but here the idea is that without getting into that mathematics we should assume that rand function can be used to carry out our experiment. Indeed the function is carefully implemented by its authors and indeed it can be used for this experiment. Still we can test the function that whether it generates sufficiently random numbers or not. In principal, this test is also required by the experimenter above, to test whether the coin he is using is fair or not. Let's see how we can do that test fairly easily that would have taken days, and convince ourself about the validity of our simulated coin.

Matlab code

```
% This script simulates a fair coin and plots its
various results.

N=1000000;
H=0;
T=0;

headbytail = zeros (1,N);

for n=1:N
    k=rand;
    if(k<0.5)
        H=H+1;
    else
        T=T+1;
    end
    headbytail(n) = H/n;
end

plot(1:N,headbytail,'LineWidth',1.5);
hold on;
plot(1:N,0.5);
ylim ([0 1]);
title('Ratio of heads')
```

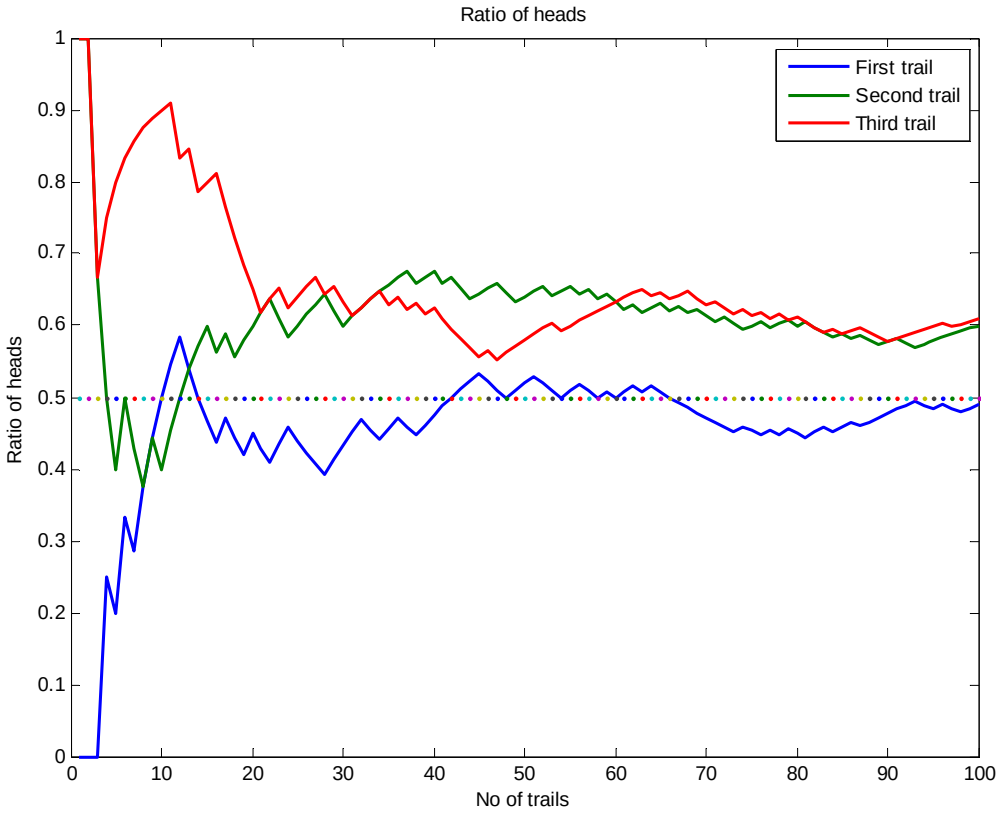
Result



You can see that ratio of heads in total is perturbing at the start but eventually stabilizes to 0.5 as the coin flips is large and is limiting to 0.5. The result of the ratio of head after the last trail is found to be 0.4998, which is fairly close to 5.

So we have seen that the simulated coin that we generated is sufficiently fair. This means that the random number generator is sufficiently random for our purpose. Although we tested it out but the test is in principal not needed because the simulated coin that we used is the same (programmed) that everybody is using for ages and has been tested by many. But in case of an experimenter's fair coin he will have to test that every time because he never knows that the coin he has is fair or not.

Now having convinced, we can carry out our actual experiment of flipping the coin 100 times. This is fairly simple, simply change the value of N in the above experiment to 100 and get the results. Let's try out the experiment a couple of times.

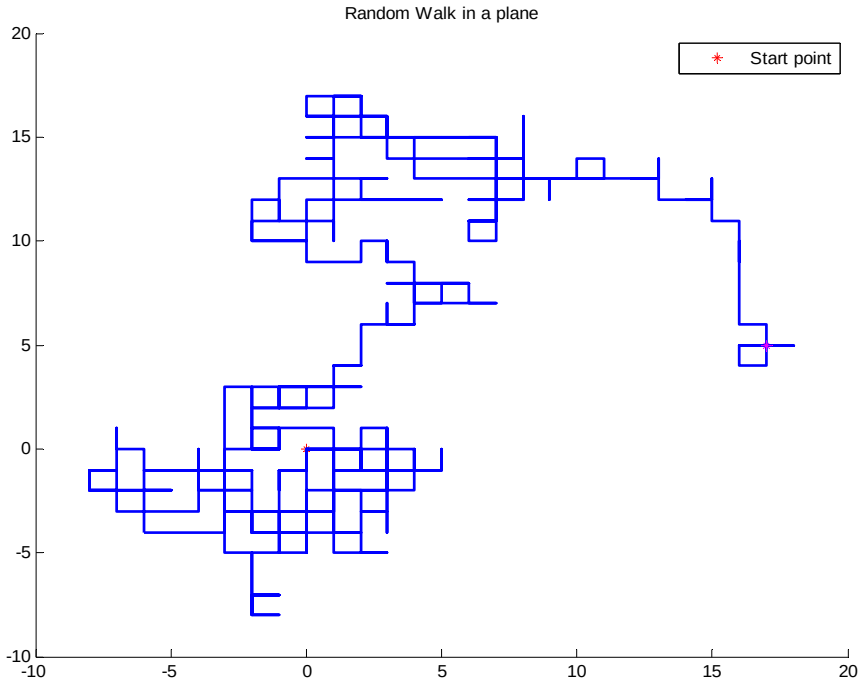


I have performed the experiment three times and potted the result; you can see that it's very erratic and quite different and random each time. The result is not shocking, this proves that even a fair coin will not give identical or equal results if flipped a small number of times, very obvious for one who understand probability theory well. You can simple change N to 1000 and get the results if you want. Note that it is impossible to tell from the output alone whether the experiment is simulated or real. So both are equivalent, you can do either way. In simulation you can get same results fairly easily. Moreover you have more power to analyse the data better in many forms including pictures and graphs which helps to draw conclusion even more easily.

Random walk

A random walk is a mathematical formalisation of a trajectory that consists of taking successive random steps. It is typically demonstrated by an example of a drunken person moving randomly on floor, so is the *'ransom walk'*. Although random walks were originally used in particular for walks that are random by a drunken or a blind person, this concept has been generalized and is now used to simulate a lot of real life situations like path traced by a molecule as it travels in a liquid or a gas, the search path of a foraging animal, the price of a fluctuating stock and the financial status of a gambler, etc. It is also used to model to model the movement of microorganisms and evolution, as the mutation is a kind of random event.

The following figure shows how a typical random walk looks like. Although this figure does not convey much about random walks but it gives us some idea about the randomness of the random walk in general. This figure can be easily generated using matlab by modifying the code that I presented in the following section.



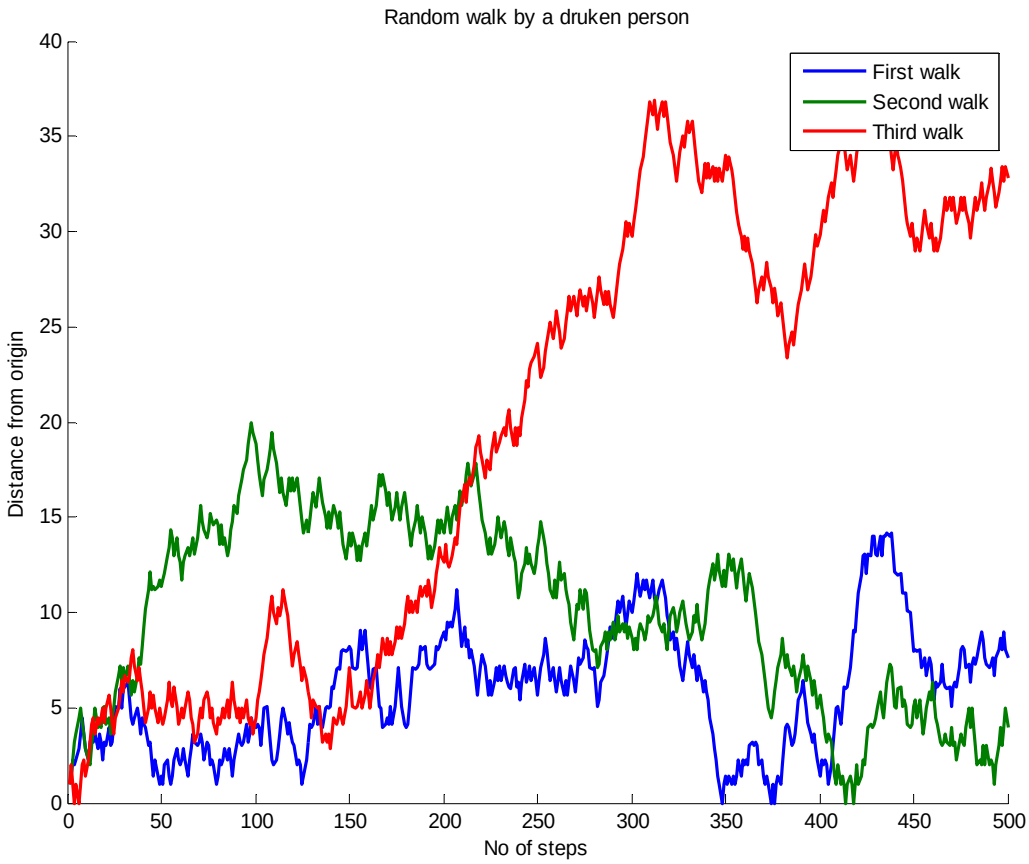
To visualize and understand this lets perform a computer experiment. In this example let's consider a drunken person moving on the plane, who can take a step randomly in any of the four directions. We are going to calculate how far he will be from origin after say 500 steps. This is expected difference from the origin and we will perform the experiment three times and visualize the results in a graph.

Matlab code

```
close all;
clear all;
clc

N=500;
L=3;
x=0;
y=0;

exp_distance = zeros(1,N);
```

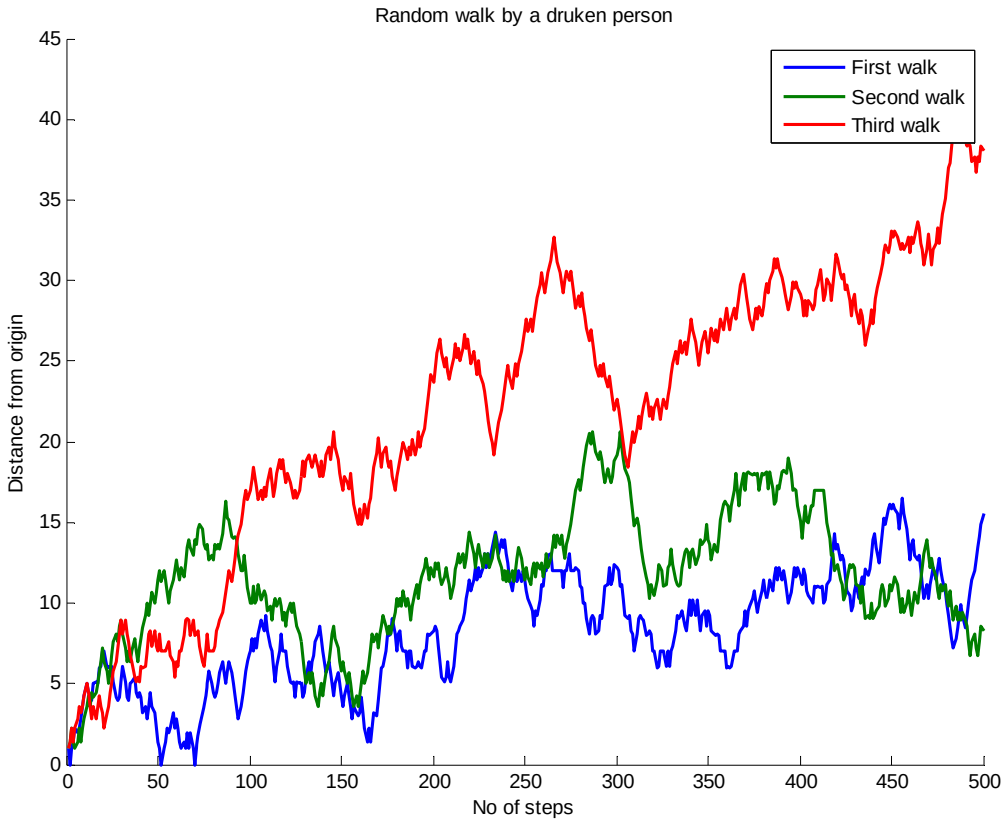
```
for l = 1:L
    x=0;
    y=0;
    % This is one single experiment simulating 100 coin
flipps
    for n=1:N
        k=rand;
        if(k<0.25)
            % consider it as west
            x=x-1;
        elseif (k<0.5)
            % consider it as north
            y=y+1;
        elseif (k<0.75)
            % consider it as east
            x=x+1;
        else
            % consider it as south
            y=y-1;
        end

        % Calculate expected distance of the drunk from
origin in each of the experiment.
        exp_distance(l,n) = sqrt(x.^2 + y.^2);
    end
    % End of the experiment.
end

% Distance from origin
figure
hold on;
title('Random walk by a druken person');
xlabel ('No of steps');
ylabel('Distance from origin');
plot(1:N,exp_distance(1,:),'LineWidth',1.5);
hold all;
plot(1:N,exp_distance(2,:),'LineWidth',1.5);
plot(1:N,exp_distance(3,:),'LineWidth',1.5);

legend('First walk','Second walk','Third walk');
```

The result



The results seems to be quite random, in the sense that one cannot point out that after say 420 steps what will be the expected distance. In the three experiments first says, it will be something around five, second says it should be close to 0 and third says it should be close to 25, so which one to believe?

So again it is quite evident that we can't perform one single experiment and believe the result. But still there is something that we can infer from the figure. If you look closely you can observe that the expected distance from the origin generally increases over time. Even in the data that we observed, if we take average the result if greater than 0. Let's try it once more.



Again looking at the data, this inference can be made that the longer the drunken person moves, the more far he seems to end at. This is a very important result that we have got from computer experiment and simulations. Think for a second if we try to infer this without any computer by performing actual experiments. So the importance of a computer simulation of a random process is very evident.

Here it is obvious that we can't believe on the 'distance from the origin' based on only one experiment. Every time you run the experiment there is a different answer of the distance. But we can have something out of this data, because of the computers at our disposal. We can perform some statistics on the data and find out what is the most typical one or we can find the average of say 10000 such experiments and say that figure as the *expected distance*. Further you will find out that as you will increase the number of trials of experiments you figure of expected distance is likely to be come out as a constant one, which you can safely assume as the real expected distance that the drunken is expected to be at from origin after x number of steps. When you do something like this, you are probably somewhere into *monti-carlo* simulations.

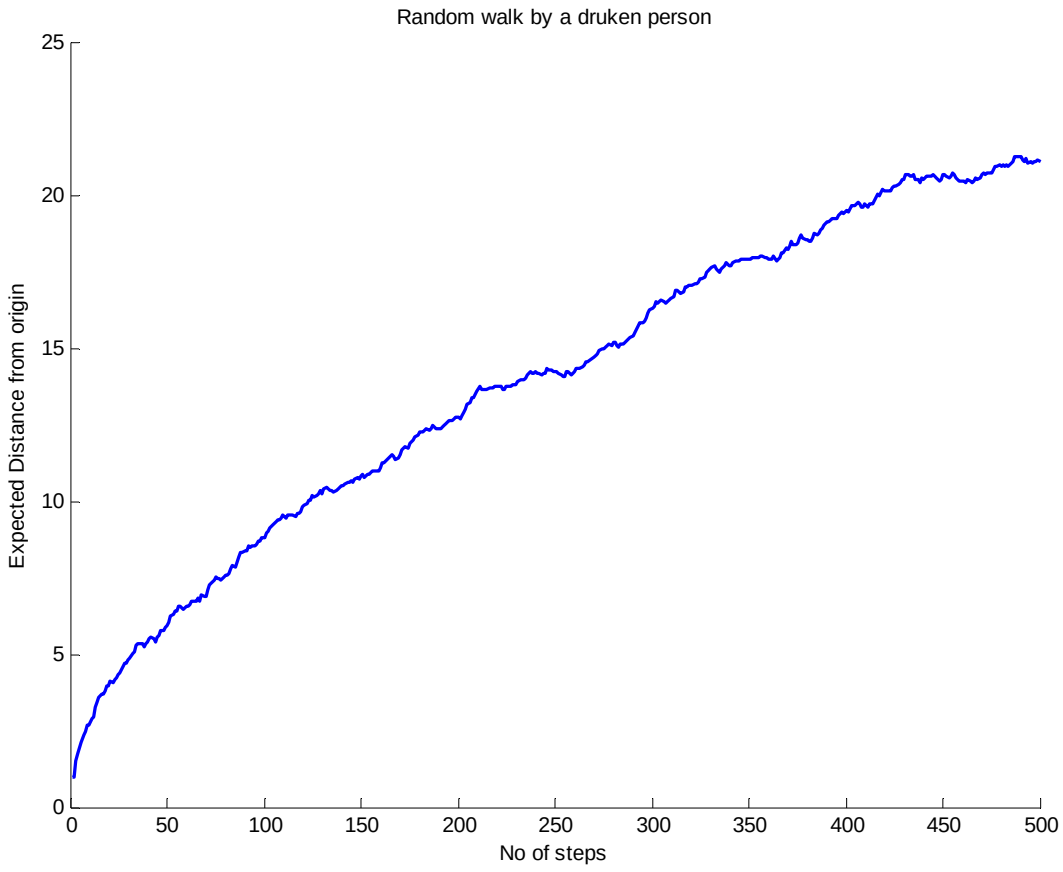
Matlab code

```
close all;
clear all;
%clc

N=500;
L=1000;
x=0;
y=0;

distance = zeros(1,N);

for l = 1:L
    x=0;
    y=0;
        % This is one single experiment simulating 100
coin flipps
    for n=1:N
        k=rand;
        if(k<0.25)
            % consider it as west
            x=x-1;
        elseif (k<0.5)
            % consider it as north
            y=y+2;
        elseif (k<0.75)
            % consider it as east
            x=x+1;
        else
            % consider it as south
            y=y-1;
        end
        distance(l,n) = sqrt(x.^2 + y.^2);
    end
    % End of the experiment.
end
exp_distance_mean = mean(distance);

% Expected distance from origin
figure
hold on;
title('Random walk by a druken person');
```

```
xlabel ('No of steps');
ylabel('Expected Distance from origin');
plot(1:N,exp_distance_mean,'LineWidth',1.5);
hold off;
disp(exp_distance_mean(N));
```

The Result

- Using 500 trails



Output

20.5264

20.1582

21.3877

20.1603

20.7621

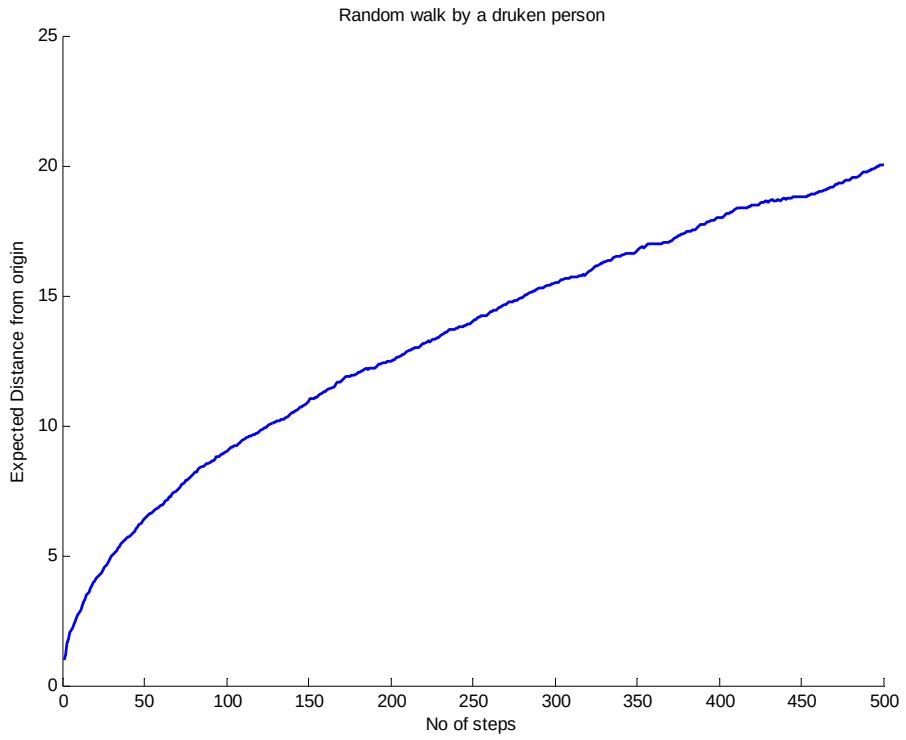
21.2915

19.8661

19.4860

Both this output and the other is the *expected distance* from the starting point, where the drunk will end up into. Running the program several times I have found these outputs, which are pretty close to 20.

- Using 1000 trails



Output

20.0901

19.8707

20.1146

20.2806

19.2497

19.7505

20.2748

20.7393

So you can easily observe the effect that I have discussed above using graphs and output data. It is evident that as we increase the trails the more realistic value of expected distance is likely to come out with very less divergence from the actual value. So the actual value is somewhere in the range of this data, which eventually come closer and closer to actual expected value as we increase the number of trails. This is used in *monti carlo* simulations in the estimation of value of pi, further in this report.

Biased random walks

Biased random walks is just like a random walk but the movements of the drunken person in different directions is not equally likely to happen, and is not uniformly distributed, instead they are biased. For that we can consider a very interesting situation in the previous drunken person example. Imagine that his drink is placed on a table and he can see that. So despite the fact he can't control his movements perfectly, he will be attracted to that particular direction. So it's more likely that he will try to move in that particular direction with some biasness in his random walks. This is a example of a 'biased random walk'.

To illustrate what differences biasness can create, we will discuss two types of biased drunks and compare the results with our normal drunk. The first one will be biased to move north, in the sense that he moves twice as much fast as he moved otherwise, looking at the glass of drink in that direction. The second drunk can move in only east or west, you can imagine him being stuck in two parallel walls.

These effects can be produced by making small changes in the matlab code that I presented in the case of normal drunk. Here are those changes

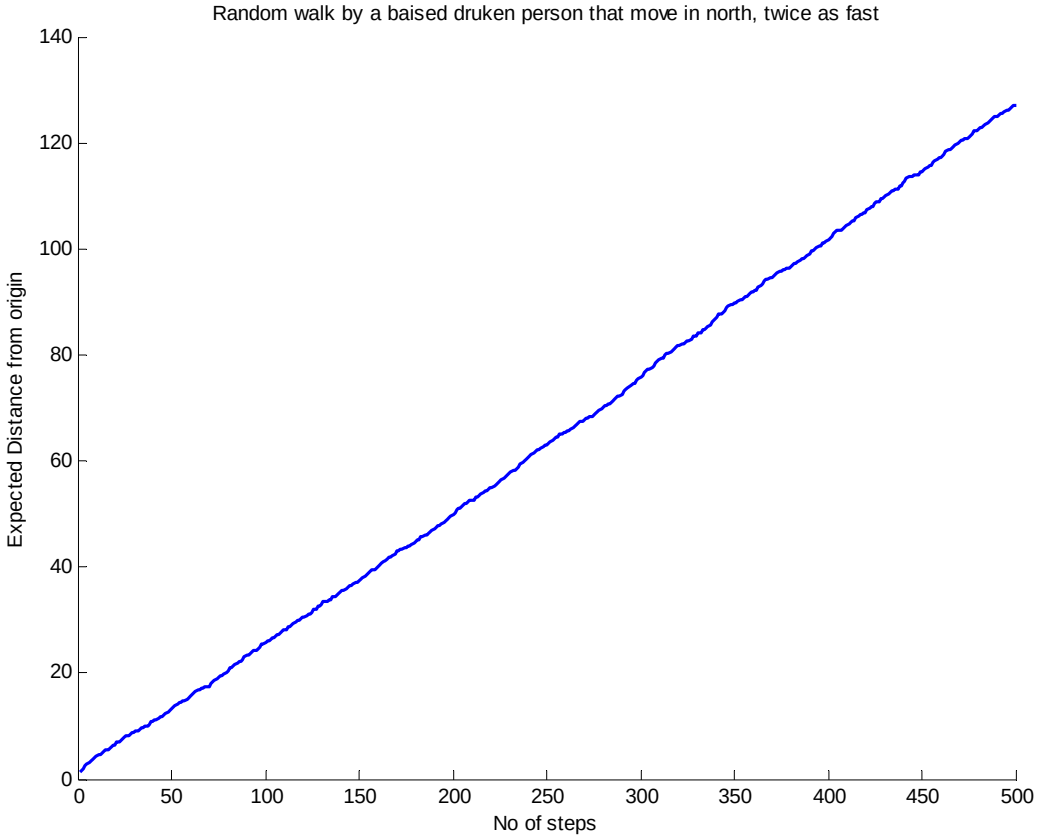
North biased drunk

```
for n=1:N
    k=rand;
    if(k<0.25)
        % consider it as west
        x=x-1;
    elseif (k<0.5)
        % consider it as north
        y=y+2;
    elseif (k<0.75)
        % consider it as east
        x=x+1;
    else
        % consider it as south
        y=y-1;
    end
    distance(l,n) = sqrt(x.^2 + y.^2);
end
```

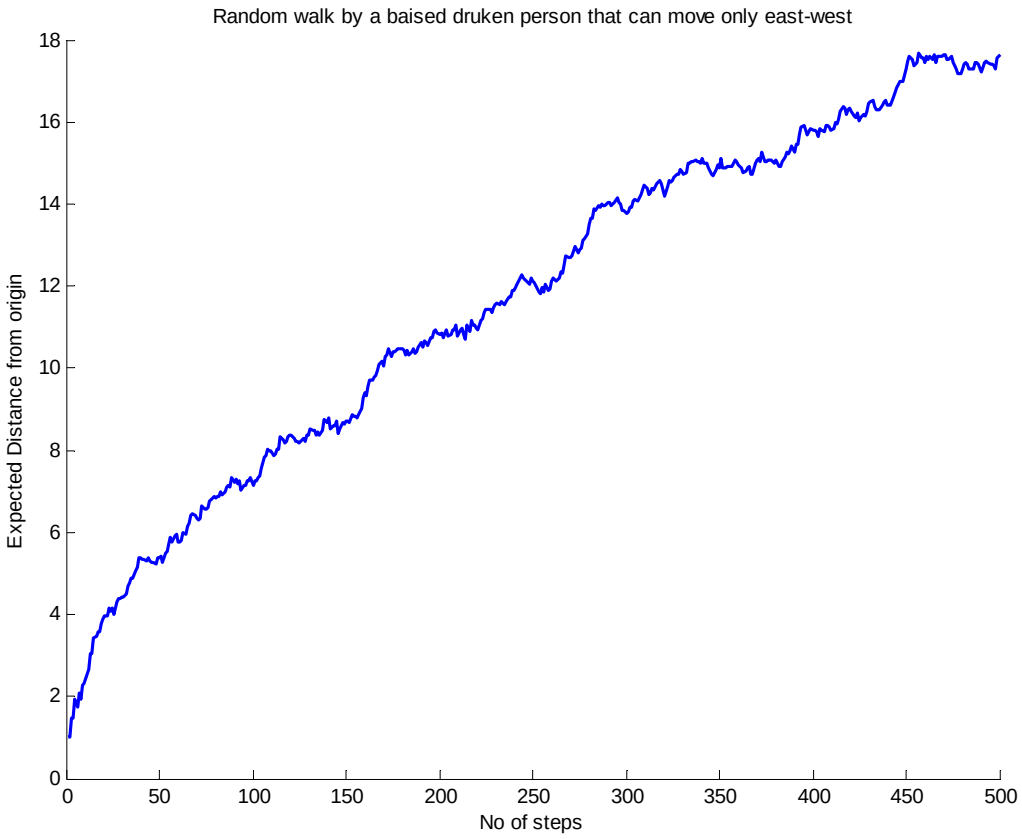
EW Drunk

```
for n=1:N
    k=rand;
    if(k<0.5)
        % consider it as west
        x=x-1;
    else
        % consider it as east
        x=x+1;
    end
    distance(l,n) = sqrt(x.^2 + y.^2);
end
```

Results



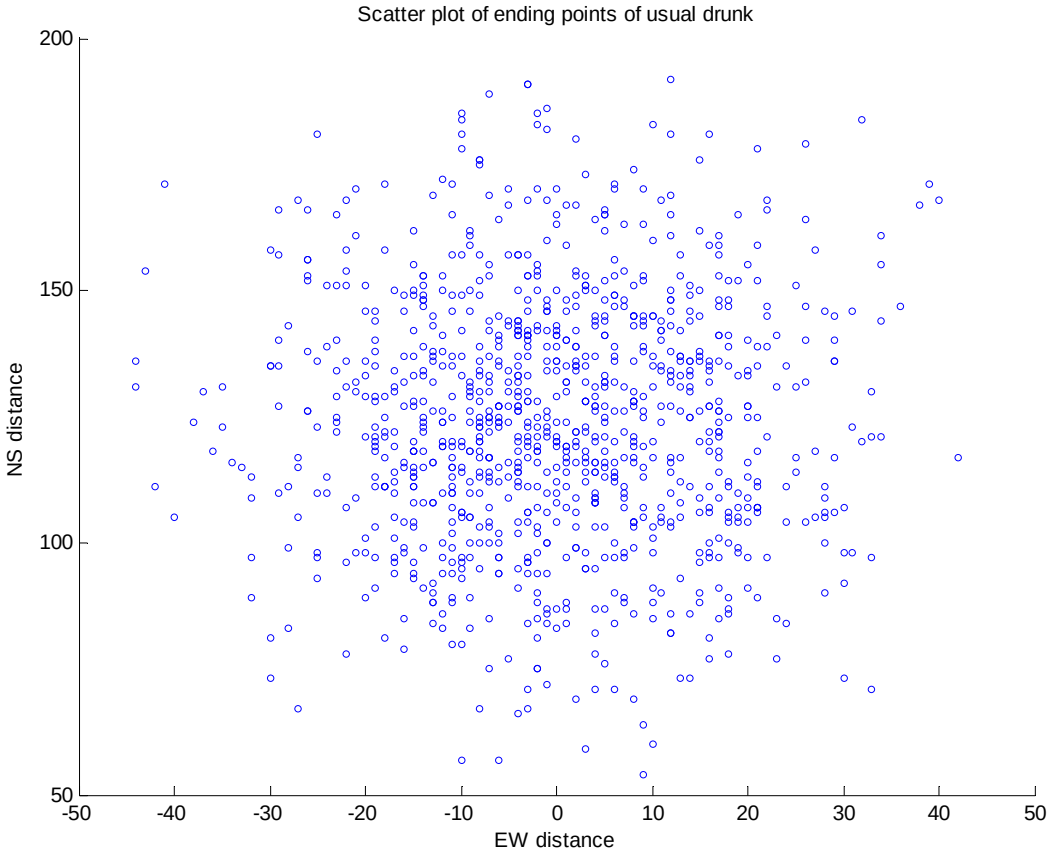
This curve is produced using 100 trials. The over smoothness of the curve is due to scale, the randomness is not just visible due to larger scale of this figure as compare to usual drunk. An important thing worth noting is that a small bias in the randomness of the drunk has produced large change in results over time. If you note it carefully, the expected distance is now close to 140 which is almost 7 times the previous result. This is main reason that casinos do so well in business. They can bias the dice a little bit and can make huge profits over time.



This is the result of east-west drunk. There is a reason for us to wonder on the result, if we compare it with usual normal drunk. Yes, the distance is almost the same compared to usual drunk, only a bit less. So this is a result to ponder, which we might have not expected.

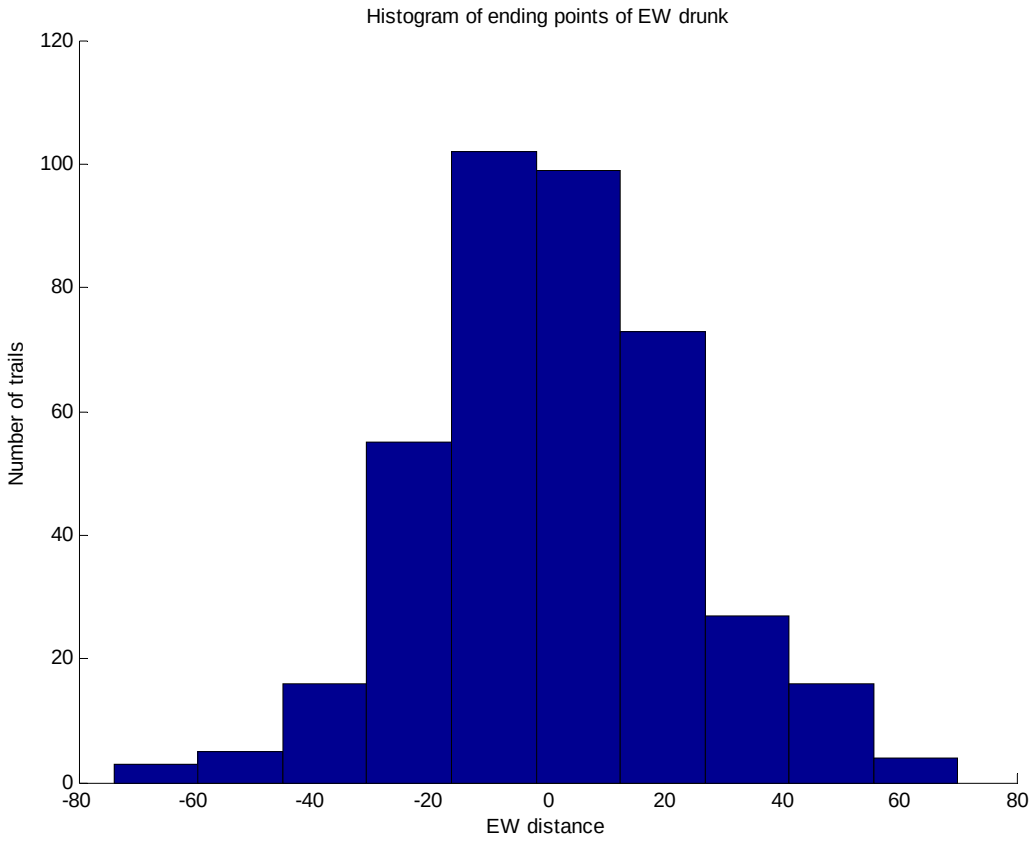
Random walks: Some more things to analyse

Computes are very efficient in handling large amount of data. Another important benefit of computer simulation is that you can present the data in many ways to conclude other things about it. Suppose we want to know how many drunks ended up where on the plane in our L trails of N walks or to identify that how many were close to average (expected difference) and how many were outlawers, and by how much amount. It's an important thing to analyse because if you actually have a drunk who is really random, you can't expect him to end up at the expected distance. So you must also get an idea about much deviation can be in the position of drunk. This thing is similar to the fact that in a head biased coin you can't be sure of a head if you do a trail. So a lot of similar statistics can be performed on the data to analyse it perfectly.



So as we observe, we get still more information from the data using some different representation which is almost impossible to draw from our previous plot using averages. It seems to be pretty much symmetric about origin, the starting point, owing to an unbiased walk. In case of a biased walk, we can expect it be a skewed in one direction. Another thing worth nothing is that the density of points in the centre seems to be more than areas, as we move away from origin.

Let's present this information a bit more clearly in the following plot



This curve is for EW drunk, similar information can be extracted; quite clear that distribution of end points is more near the origin, starting point that far away. Another interesting conclusion that we can draw is that the curve seems like a normal or close to normal/Gaussian distribution.

The two examples further prove and illustrate the benefits of a computer random experiment simulation. Things that you may conjecture in random events and that common sense says may often not likely to be correct, when it comes to random events. So computer simulations provide a very efficient way to find out results and make interesting inferences about them and later explaining those results.

Monti carlo simulations

In random walks, where we performed many trails of the same experiment and then calculated and plotted expected distance that represents all the trails in one single curve, we already have seen what a monti carlo simulation is. In this portion, I would like to focus on other interesting problems that can be solved by monti carlo method.

Monti carlo simulations, essentially is based on inferential statistics. It is an application of inferential statistics which assumes an important property to be true, that random sample tends to exhibit same properties as the population from which it is drawn. We choose a random sample chosen from the population and we assume that it represents the population and exhibits the same properties.

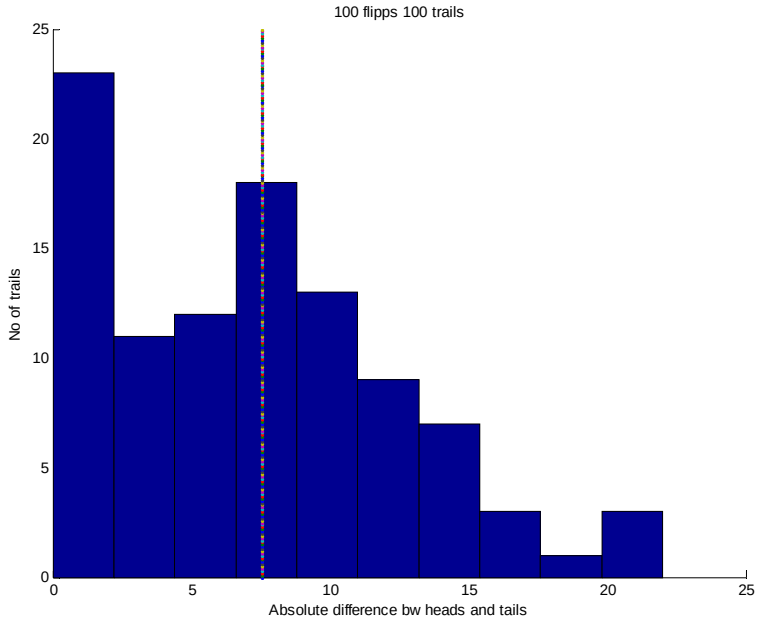
Let's consider an example that I have already discussed in the beginning of the topic once again, the coin flip where we are going to check more effectively and quantitatively whether the coin is fair or not. For that we can repeat the experiment and perform a bunch of trails of the experiment (100 flips) and plot the results. We will have a closer look on the head-tails and head/tail ratio of a coin and will consider how it changes with the number of flips and trails of the experiment.

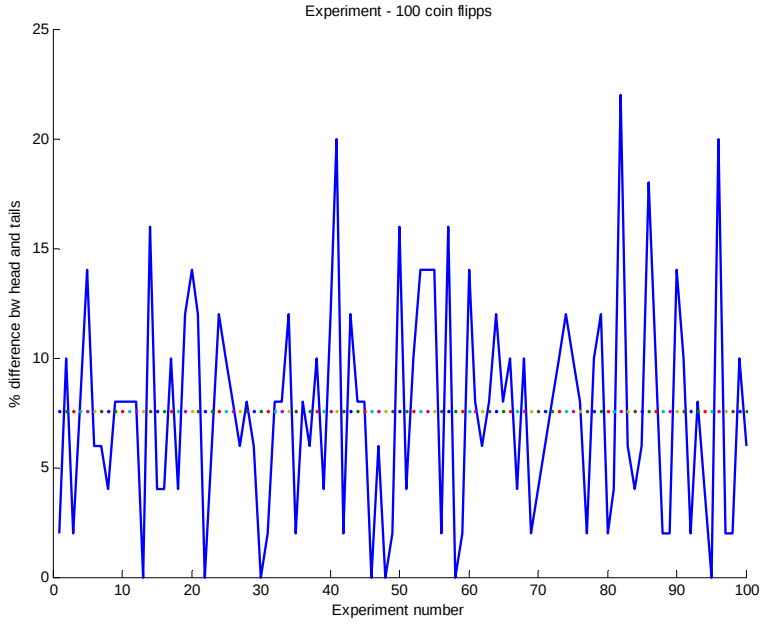
Matlab code

```
% This script simulates a coin flip 100 times and we
can do N trials of the
% experiment

% Experiment - 100 coin flipps
% No of Trails - N

close all;
clear all;
clc

N=100;
H=0;
T=0;

headbytail = zeros(1,N);
headminustail = zeros(1,N);
```

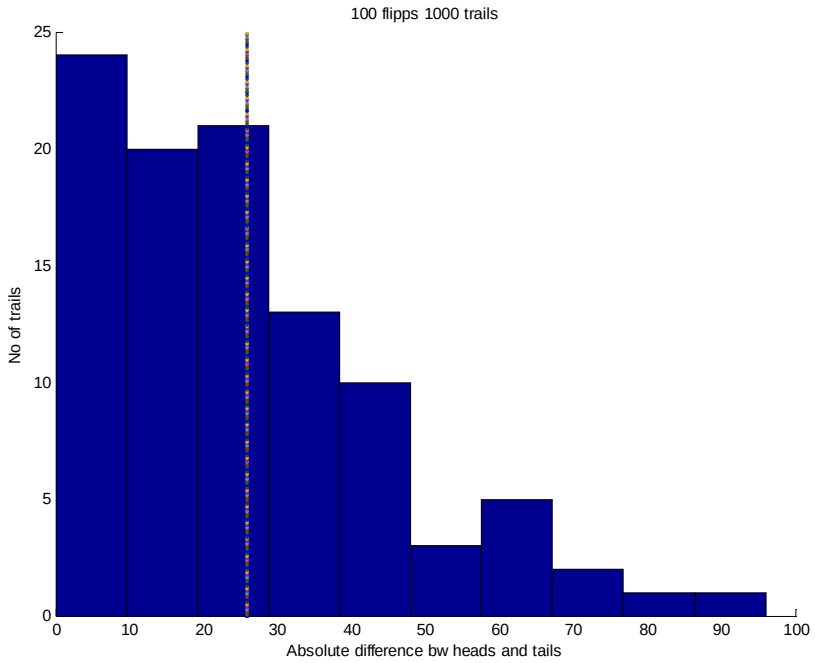
```
for l = 1:N
    H=0;
    T=0;
    % This is one single experiment simulating 100 coin
flipps
    for n=1:100
        k=rand;
        if(k<0.5)
            H=H+1;
        else
            T=T+1;
        end
    end
    % End of the experiment.
    % We calculate H/T and H-T for each experiment.
    headminustail(l) = abs(H-T);
    headbytail(l) = headminustail(l)/(H+T)*100;
end
figure
hold on;
title('Experiment - 1000 coin flipps');
xlabel ('Experiment number');

ylabel('% difference bw head and tails');
plot(1:N,headbytail,'LineWidth',2);
plot(1:N,mean(headbytail));
hold off;

figure
hold on;
title('100 flipps 1000 trails');
xlabel('Absolute difference bw heads and tails');
ylabel('No of trails');
hist(headminustail);
plot(mean(headminustail),0:0.1:25)
hold off;
```

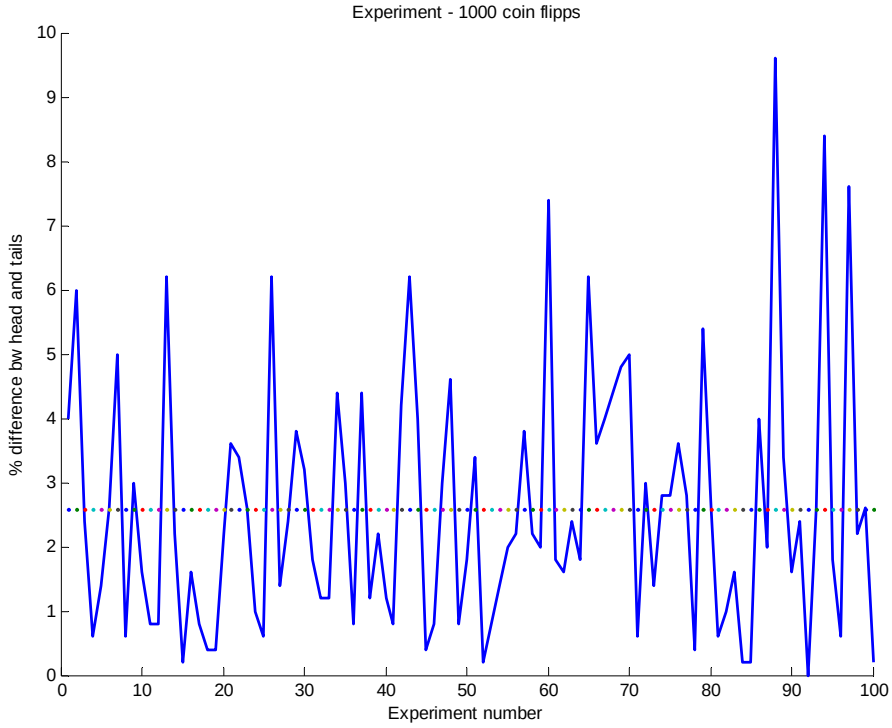
The results





This plot represents the percentage difference of head and tails. So in essence it normalizes the data presented in the histogram. The importance of this plot is that when we increase the number of trails our difference between head and tails may increase but the percentage difference may still decrease. In essence it's the percentage difference that we are generally more interested in.


100 flipps 1000 trails
No of trails
Absolute difference bw heads and tails
0
5
10
15
20
25
0
10
20
30
40
50
60
70
80
90
100


Experiment - 1000 coin flipps
% difference bw head and tails
Experiment number
0
1
2
3
4
5
6
7
8
9
10
0
10
20
30
40
50
60
70
80
90
100

If we compare the graphs of of 100 and 1000, we observe that in 1000 the difference between heads and tails, the mean difference has increased. This is due to more number of flips but the percentage difference in 1000 has decreased. Moreover consider the range of percentage difference in 100 and 1000, in 1000 flips plot, the range is from 0 to nearly 8 whereas in its counterpart in 100 flips plot, the range is from 0 to 25. This is a very important result, in the sense that it proves by flipping more coins, experiment becomes more *reproducible*. The range is narrower and each experiment tends to give an answer that is closer to all the other experiments. This makes us believe that the answer that we may get in future by a similar experiment is also between this range and you can observe the output values as we increase the number of flips we will observe that the range will be very tight.

Now instead of plotting the graphs, we can display the mean of all the trials of the experiment. In other words, in each experiment we get a head/tail ratio, we can take the mean of that and display the result in matlab. The same can be achieved with slight modification of above code, which is by using this statement

```
mean (headbytail)
```

Output

For 100000 flips

ans =

0.2290

0.2393

0.2841

0.2569

So as you expected the more the number of flips, the percentage difference should approach 0. Lets see what happens if we flip coins 10 times than this amount.

For 1000000 flips

ans =

0.0894

0.0718

0.0797

0.0800

So as we expected the answer is pretty close to 0 and you can observe the range of values obtained by looking at the data, it is also very tight.

Let's perform another computer experiment of flipping 50 coins. Suppose we are interested in knowing what is expected to be the result (diff of heads and tails) of flipping a fair coin 50 times. Of course if you run the experiment one single time, you will get an answer and if you run the experiment a million times still you will get an answer. But by now you are quite sure that one single or even a few times running the experiment may not give the answer that is actually expected answer. So let's analyse the output of program by changing the number of trails. This statement will do the job.

```
disp(mean(headminustail))
```

Output

For 10 flips

5.4000

5

5.2000

3.8000

4.8000

3.6000

8.2000

The answer is in points because we are calculating the mean. You can see that the answer is very fluctuating and the range of answer is large. Consider 1000 flips below.

For 1000 flips

5.6480

5.4940

5.3720

5.5520

5.5220

Yes the output is now less fluctuating and appears to be settling down. Although the coin is fair but the output doesn't seems to be close to 0. This result seems to be astonishing. Let's consider a 1000000 flips to study, if the answer might change.

For 1000000 flips

5.6195

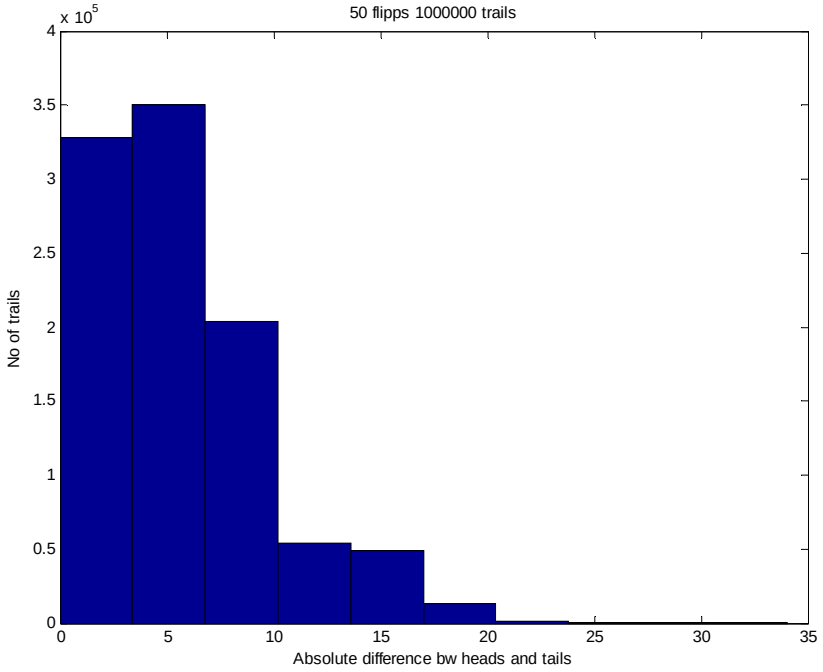
5.6167

5.6035

5.6147

5.6149

Using 1000000 flips, we are quite sure that answer is not going to change or shift anywhere. This is the expected difference in heads and tails if we flip, a 50 coins. But should be expect and infer as per monti carlo simulations that if we flip a 50 coins the answer will come out to be around 5.6. Before making any such prediction have a look at the histogram here

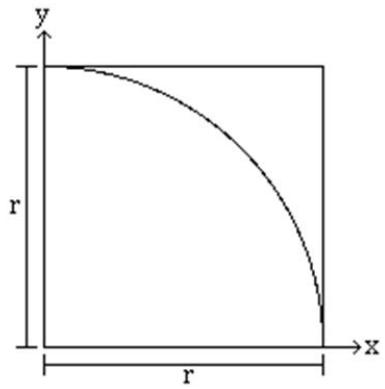


This figure shows that your result can be anywhere in between from 0 to 20 with the corresponding probabilities which you can calculate from this figure. This figure remains constant even for higher values of trails. If you calculate the expected value from the probabilities the answer is going to be 5.6. This is the most practical result of 50 flips that can be concluded.

Estimation the value of pi

This is a very interesting simulation in which we can estimate the value of pi, quite accurately using only random processes. The solution is interesting in the sense that, we can estimate a quantity that has nothing to do with randomness using random process simulations.

Consider a quarter circle inscribed in a square, equal to its radius as shown below.



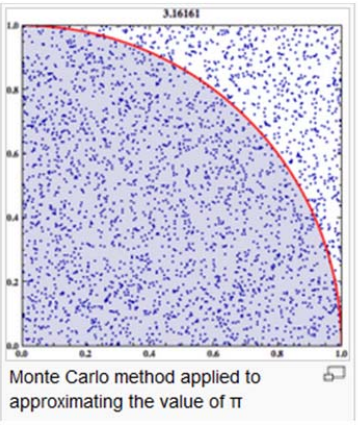
Now we randomly throw darts or draw random points, in the space covered by the square. Some of these darts that we assume to be hitting at random locations, will hit inside the circle while others will land outside the circle. As we have assumed that these darts are falling really randomly, we can expect that after throwing a large number of darts, the ratio of darts hitting inside the quarter circle will be propotional to ratio of area of quarter circle to the total area.

$$\frac{\#\text{ darts hitting shaded area}}{\#\text{ darts hitting inside square}} = \frac{\frac{1}{4}\pi r^2}{r^2} = \frac{1}{4}\pi$$

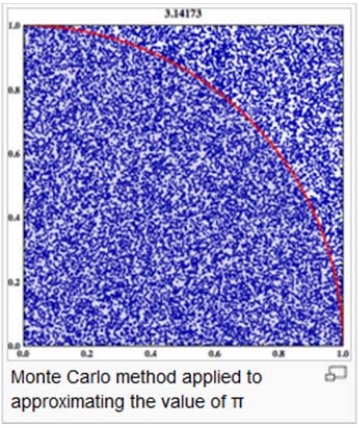
or

$$\pi = 4\,\frac{\#\text{ darts hitting shaded area}}{\#\text{ darts hitting inside square}}$$

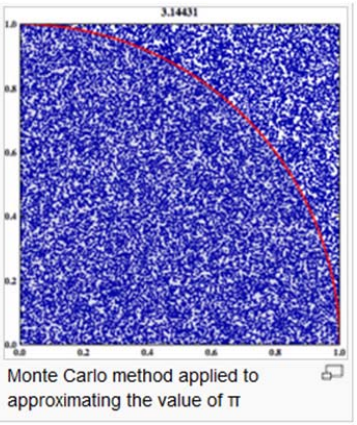
So this simple algebra relates a deterministic value to the random process and gives us the estimation of the value of pi. Here is how a the situation look like over time, when we throw darts to estimate the value of pi.



Monte Carlo method applied to approximating the value of π



Monte Carlo method applied to approximating the value of π

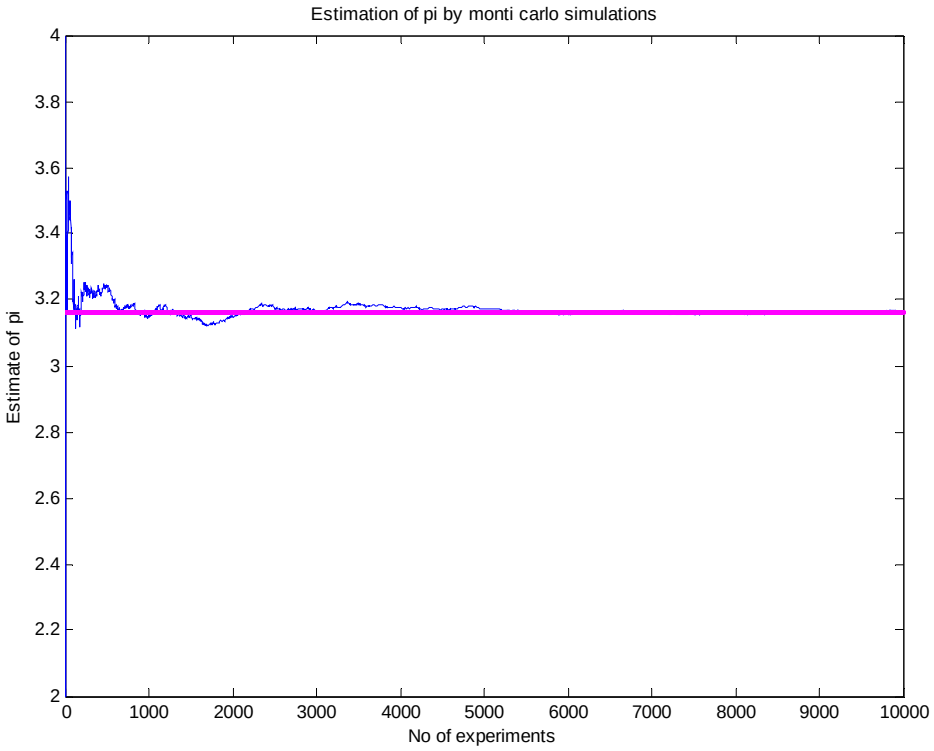


Monte Carlo method applied to approximating the value of π

Code

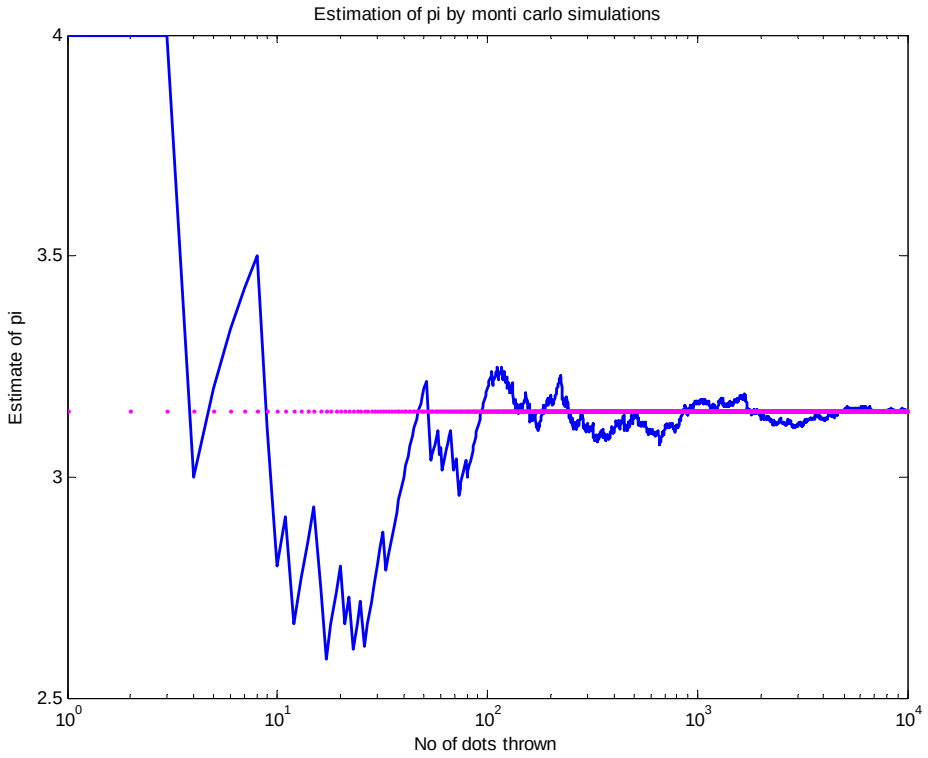
```
% Monti carlo simulation to estimate the value of pi

clear all;
close all;


N = 100000000;
inCircle =0;
pi_e = zeros(1,N);

for n = 1 : N
    x = rand;
    y = rand;

    if (x*x + y*y < 1)
        inCircle=inCircle+1;
    end

    %calculate pi
    pi_e(n) = 4 *inCircle/n;
end

disp (['The estimate of pi is ' num2str(pi_e(N))] );

%figure
%semilogx(1:N,pi_e,'LineWidth',1.5)
hold on;

title('Estimation of pi by monti carlo simulations')
xlabel('No of dots thrown');
ylabel('Estimate of pi');
plot(1:N,pi,'m')
```

Results



On the y axis we have the value of pi, obtained by throwing N number of darts. Here we have thrown 10000 darts and the estimated value of pi obtained after 10000 darts thrown, is **3.16**. You can observe that in the initial phase there is quite lot of randomness in the value of pi but later on this randomness eventually evaporates and result seems to be settling around a value, which is actually the answer. Mathematically this is due to fact that as we increase the number of darts thrown, the ratio which is the number of darts in the quarter circle to the total darts thrown won't change much by adding a few darts to numerator or the denominator. This value of ratio settles to a value and the value of pi is proportional to this ratio will also eventually settle down.

While in the beginning of the process, when only a few darts has been thrown, this ratio will change a lot depending on the case that whether the dart falls inside or outside the quarter circle.



To illustrate the effect of perturbation at the start of the process, I have plotted the same result on a logarithmic axis (x axis), which gives us a better view of what happens in the beginning and what the process eventually end at. So you can easily visualize here, the ratio changes very randomly up to 20 darts but after 1000 darts it seems to be getting stabilized.

Now I think we have got sufficient exposure to consider the definition in Wikipedia about monti carlo simulations. I intentionally did not mention it before because, I thought we should have a better insight of monti carlo simulations before getting into the complicated definition. "Monte Carlo methods (or Monte Carlo experiments) are a class of *computational algorithms* that rely on *repeated random sampling* to compute their results. Monte Carlo methods are often used in simulating physical and mathematical systems. These methods are most suited to calculation by a computer and tend to be used when it is infeasible to compute an exact result with a deterministic algorithm. This method is also used to complement the theoretical derivations."

I would like to summarize it by stating the fundamental idea behind Monte Carlo simulation for inferential statistics is that insights regarding the characteristics of a statistic can be gained by repeatedly drawing random samples from the same population of interest and observing the behaviour of the statistic over the samples.

As you have guessed already, we can make the answer more accurate statistically, by increasing the number of darts and calculating values. For 1 million darts the estimated value of pi comes out to be 3.141816 and it's always quite the same, no matter how many times we run the program. That should give us some idea that we are pretty close to the answer that statistically is the right answer.

Magnetocaloric effect

The Magnetocaloric effect (MCE) is a magneto-thermodynamic phenomenon in which a reversible change in temperature of a suitable material is caused by exposing the material to a changing magnetic field. This is also known by low temperature physicists as *adiabatic demagnetization*, due to the application of the process specifically to create a temperature drop. Also a related term adiabatic temperature change (ΔTad), is also used interchangeably with magnetocaloric effect.

It was first discovered in 1881 by Warburg, who was investigating iron, and was later independently explained by Debye and Giauque. They also suggested the first practical use of the MCE: the adiabatic magnetization, for reaching temperatures lower than that of liquid helium, which had been the lowest achievable experimental temperature.

This effect of refrigeration obtained from magnetocaloric effect is known as **magnetic refrigeration**. With this technique temperature ranging well below from 1K to normal common refrigerator temperatures has been obtained. Due to the cooling effect obtained, this field is extensively being researched to replace conventional CFC refrigerators that pollute the environment with a cleaner and energy efficient magnetic refrigerator.

Phenomenon

MCE can be understood in a material in which its effect is prominent, that is, which is paramagnetic or ferromagnetic near its *ordering temperature*. At this ordering temperature there is a sudden change in the magnetic state of the compound - a so-called *magnetic transition*. The material is magnetic because it contains metal atoms that themselves act like tiny bar magnets. When it is warmed up from sub-zero temperatures, a point comes where these atomic magnets abruptly change the way they are lined up. This switch occurs at different temperatures when the material is placed in a magnetic field. So applying such a field can trigger the magnetic transition, and the resulting realignment of atomic magnets can then cause the material to lose heat and become colder - in other words, it shows a negative MCE.

The entropy of such a system can be considered as a sum of two contributions, the entropy related to magnetic ordering and the entropy related to the temperature of the system. Application of a magnetic field will order the magnetic moments comprising the system, which are disordered by thermal agitation energy, and, consequently, the entropy depending on the magnetic ordering (the magnetic entropy) will be lowered. If a magnetic field is applied under adiabatic conditions when any heat exchange with the surroundings is absent, then the entropy related to the temperature should increase in order to preserve the total entropy of the system constant. Increasing of this entropy implies the system heating up, and an increase in temperature. The opposite process—adiabatic removal of the magnetic field—will cause cooling of the magnetic system under consideration.

All magnetic materials intrinsically show MCE, although the intensity of the effect depends on the properties of each material. Apart from that it depends upon the magnetic field change and the temperature at which it is measured. Most magnetic materials exhibit a large MCE only at low temperatures, which is not suitable for domestic usage. But with the discovery of giant MCE (GMCE) in Gd5Si2Ge2, which is due to a simultaneous magnetic and crystallographic first order transition in 1997 by Pecharsky and Gschneidner, this field has gained a new momentum in scientific community across the world.

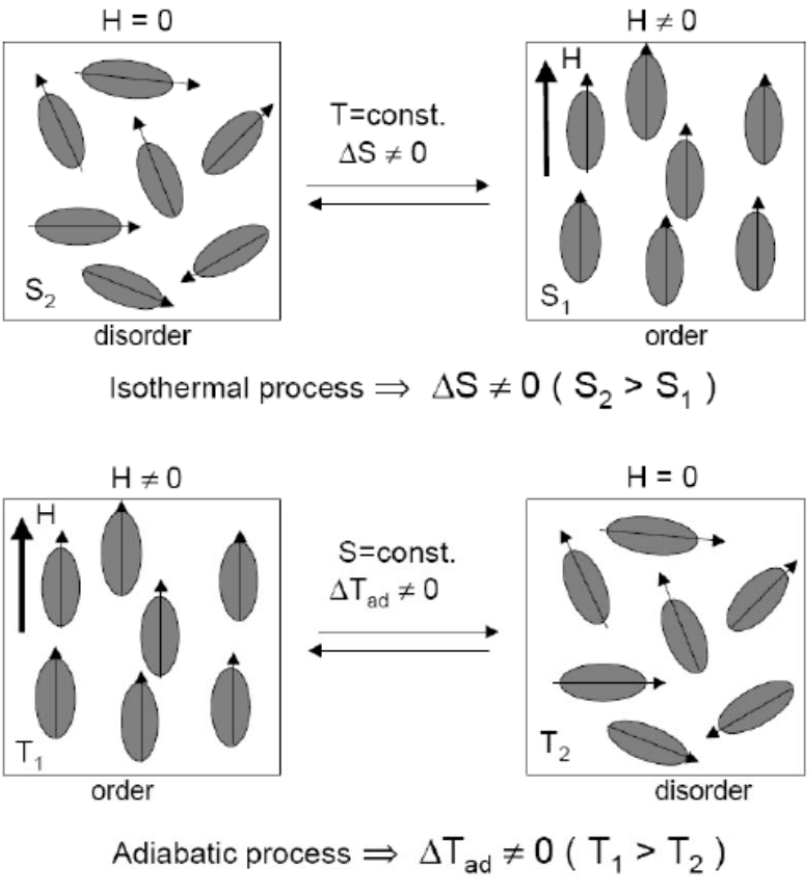


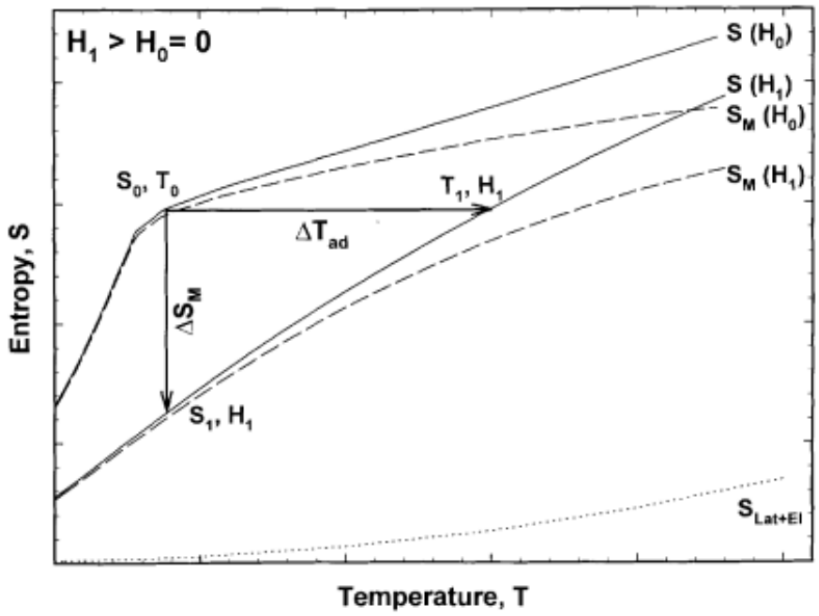
Figure 1 – Schematic picture that shows the two basic processes of the magnetocaloric effect when a magnetic field is applied or removed in a magnetic system: the isothermal process,which lead to an entropy change, and the adiabatic process, which yields a variaton in temperature.

Calculation and objective

At constant pressure the entropy of a magnetic solid S(T,H), which is a function of both magnetic field strength (H) and the absolute temperature (T), is combined total of the magnetic, S_M, lattice S_{Lat} and electronic, S_{El} contributions

$$S(T, H) = S_M(T, H) + S_{Lat}(T) + S_{El}(T).$$

The thermodynamics of the MCE in a ferromagnet near its magnetic ordering temperature (Curie temperature, T_C) is illustrated in the figure below



The S-T diagram illustrating the existence of the magnetocaloric effect. The solid lines represent the total entropy in two different magnetic fields: H_O=0 and H_1>0. The horizontal arrow shows ΔT_{ad} and the vertical arrow shows ΔS_m when the magnetic field is changed from H_0 to H_1. The dotted line shows the combined lattice and electronic (non-magnetic) entropy, and dashed lines show the magnetic entropy in the two fields. S_0 and T_0 are zero field entropy and temperature, S_1 and T_1 are entropy and temperature at the elevated magnetic field H_1.

$\Delta T_{ad}(T, \Delta H)$ and $\Delta S_m(T, \Delta H)$ represent the two quantitative characteristics of magnetocaloric effect and it is obvious that both $\Delta T_{ad}(T, \Delta H)$ and $\Delta S_m(T, \Delta H)$ are functions of the initial temperature, T_0 (i.e. the temperature before the magnetic field was altered), and the magnetic field change, $\Delta H=H_1-H_0$).

The $\Delta T_{ad}(T, \Delta H)$ and $\Delta S_m(T, \Delta H)$ are correlated with the magnetization (M), the magnetic field strength, the heat capacity at constant pressure (C), and the absolute temperature by one of the fundamental Maxwell's relations

$$\left(\frac{\partial S(T,H)}{\partial H}\right)_T = \left(\frac{\partial M(T,H)}{\partial T}\right)_H,$$

which for an isothermal isobaric process after integration and doing some further mathematics yields the following two expressions

$$\Delta S_M(T,\Delta H) = \int_{H_1}^{H_2}\left(\frac{\partial M(T,H)}{\partial T}\right)_H dH.$$

$$\Delta T_{ad}(T, \Delta H) = -\int_{H_1}^{H_2}\left(\frac{T}{C(T,H)}\right)_H\left(\frac{\partial M(T,H)}{\partial T}\right)_H dH.$$

These equations have a fundamental importance on the understanding of the behaviour of the MCE on solids, and serve as a guide for the search of new materials with a large magnetocaloric effect in which involves the search of temperature for which $\Delta T_{ad}(T, \Delta H)$ and $\Delta S_m(T, \Delta H)$ is maximum.

Measurements

Experimental measurements are carried out using either **direct measurements** or **indirect measurements**. Direct techniques to measure MCE always involve the measurement of the initial T_0 and the final temperatures T_F of the sample, when the external field is changed from an initial H_O to a final H_F value. Then the measurement of the adiabatic temperature change is simply given by

$$\Delta T_{ad}(T_0, H_F - H_0) = T_F - T_0 .$$

Other technique is **Indirect method** which involves either heat capacity measurements, that allow us to calculate $\Delta T_{ad}(T, \Delta H)$ and $\Delta S_m(T, \Delta H)$, or magnetisation measurements which allow us to calculate only $\Delta S_m(T, \Delta H)$. Magnetization must be measured as a function of T and H and by numerical integration it is said to be very useful as a rapid search for potential magnetic refrigerant materials.

The measurement of the heat capacity as a function of temperature in constant magnetic fields and pressure provides the most complete characterization of MCE in magnetic materials. The entropy of a solid can be calculated from the heat capacity as:

$$S(T)_{H=0} = \int_0^T \frac{C(T)_{P,H=0}}{T} dT + S_0$$

$$S(T)_{H\neq 0} = \int_0^T \frac{C(T)_{P,H}}{T} dT + S_{0,H}$$

Where S_O and $S_{O,H}$ are the zero temperature entropies. In a condensed system $S_O = S_{O,H}$. Hence, if $S(T)_H$ is known, both $\Delta T_{ad}(T, \Delta H)$ and $\Delta S_m(T, \Delta H)$ can be obtained. However this evaluation is not valid if a first-order transition takes place within the evaluated range, since the value of C_P *is* not defined at a first order transition.

Calculation of MCE for $TbRu_2Ge_2$

The calculation of MCE of $TbRu_2Ge_2$ is carried out indirectly by measuring heat capacities in the temperature range of about 2.5K to 295K using *PPMS VSM (Vibrating sample) magnetometer* available at the IITB magnetic materials lab. The heat capacities are calculated for 0 KOersted and 50 KOersted. The data is imported to matlab for analysis and calculation of MCE. The calculation of S_0, S_{10} and S_{50} which are entropies at corresponding magnetic field, calculated from the heat capacity data by numerically integrating the equation below using recursive adaptive Simpson quadrature method.

$$S(T)_{H=0} = \int_0^T \frac{C(T)_0}{T} dT + S_0$$

and

$$S(T)_{H\neq 0} = \int_0^T \frac{C(T)_H}{T} dT + S_{0,H},$$

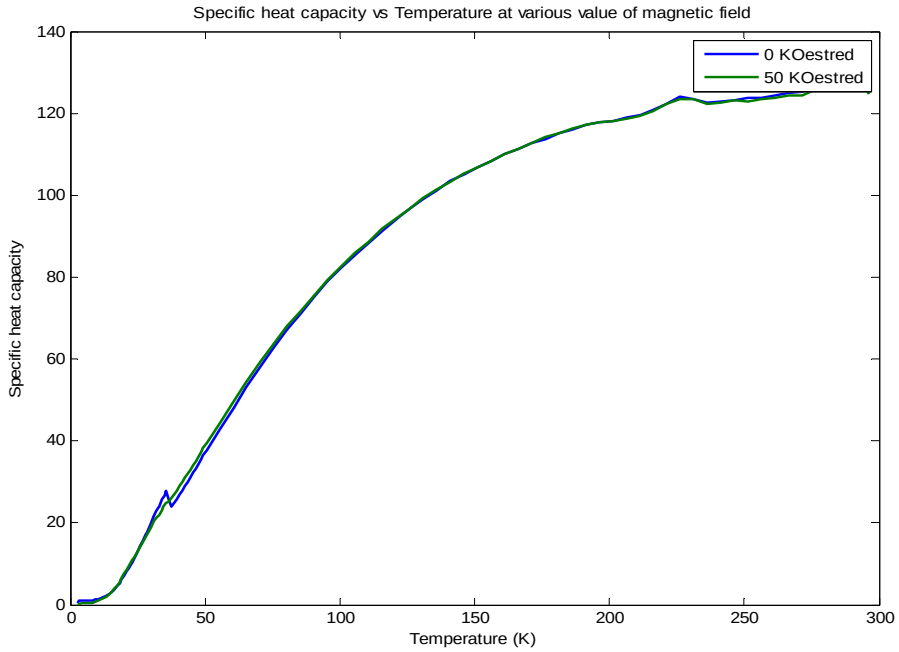
And S_0 and S_{OH} both calculated using formula

$$S_O = \frac{C_T}{2 * T_{Lowest}}$$

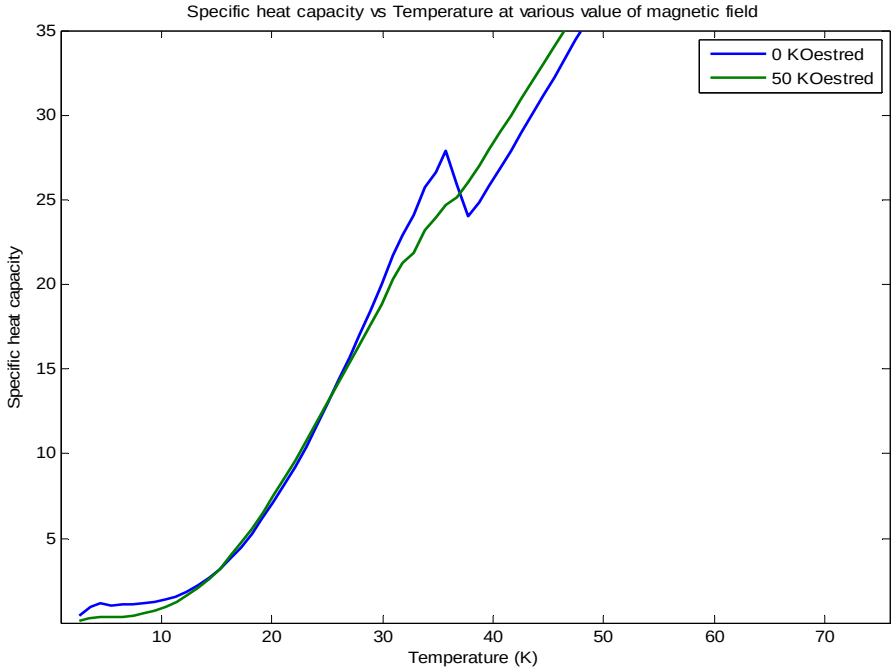
The results calculated indirectly using the data taken from magnetometer is discussed in the next topic. The results are in agreement with experimental evidence.

Results

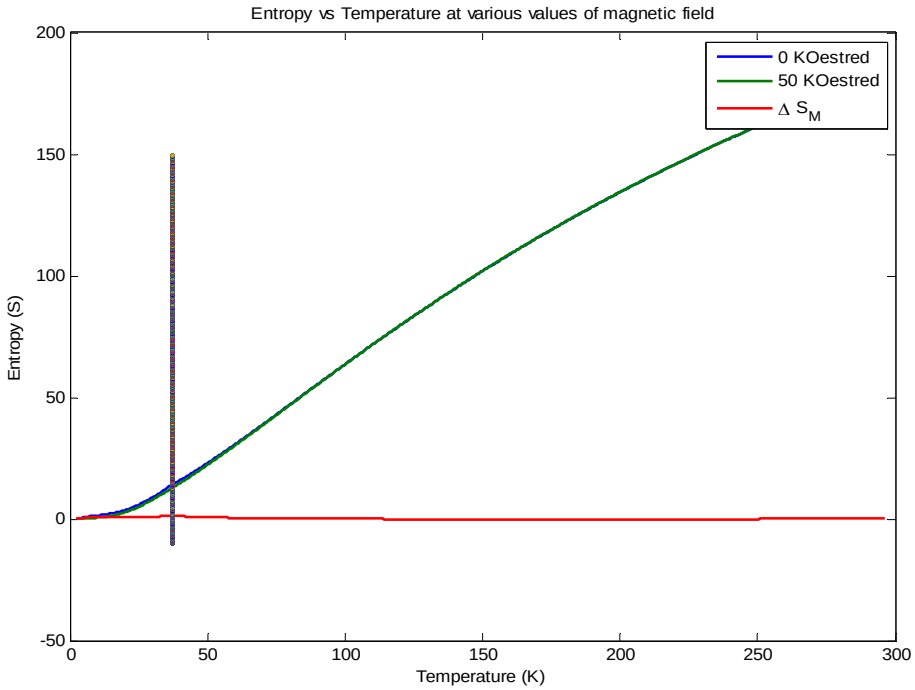
Plots of specific heat capacity



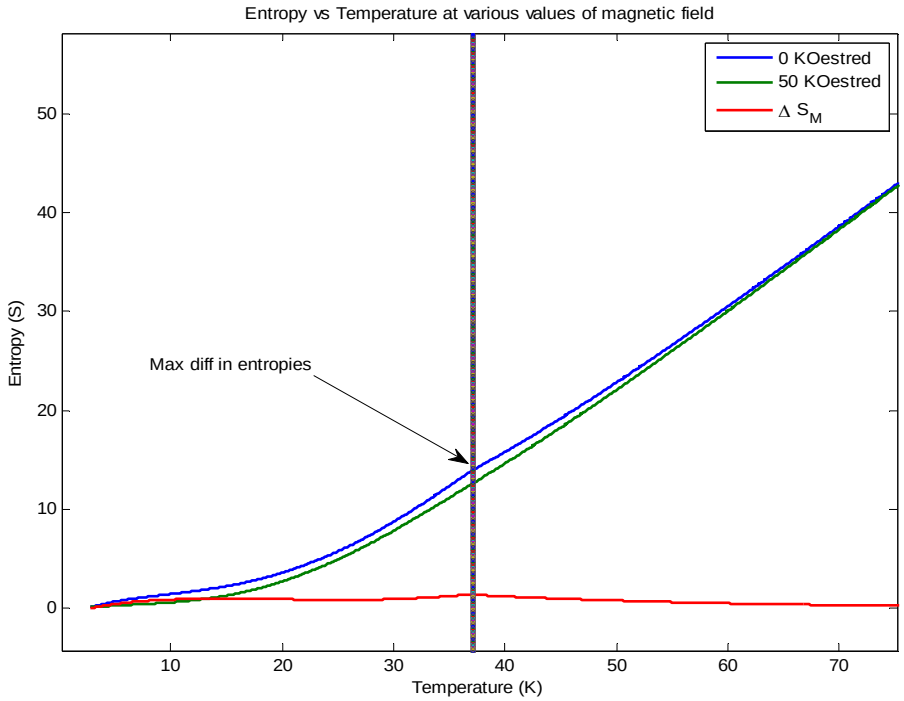
Closer look



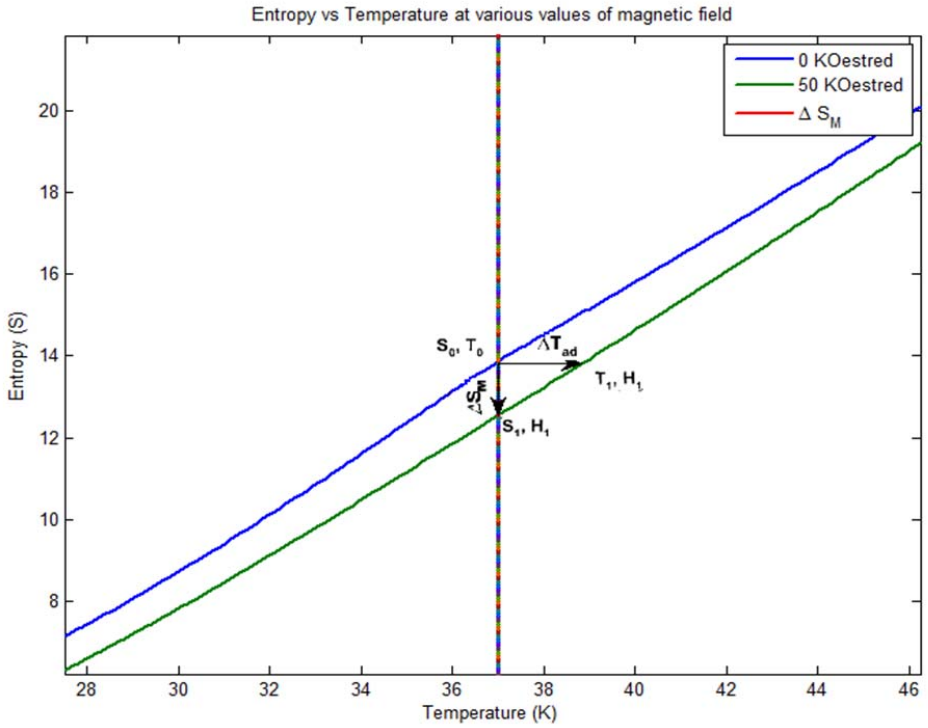
Entropies



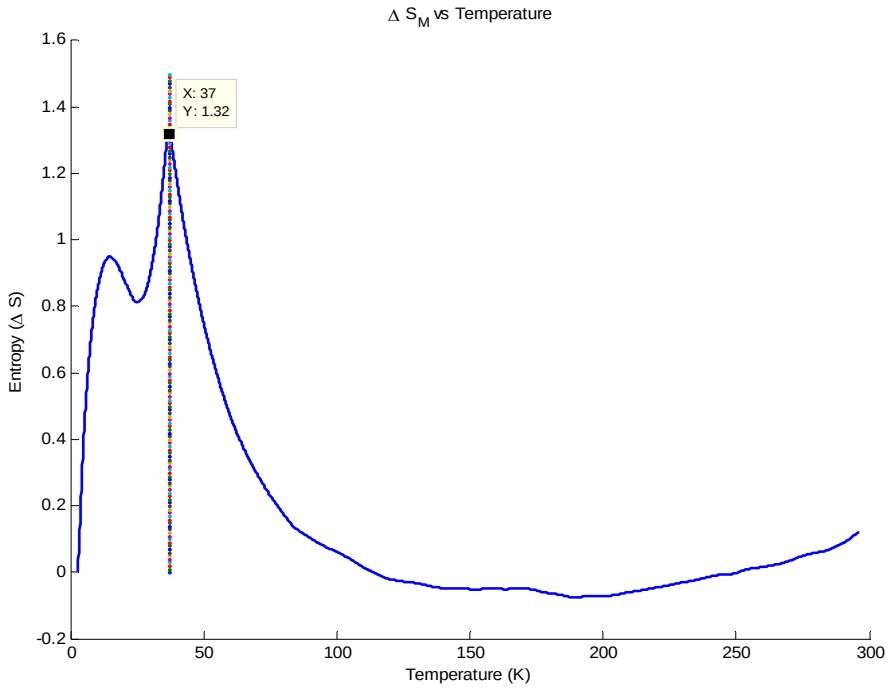
Closer look



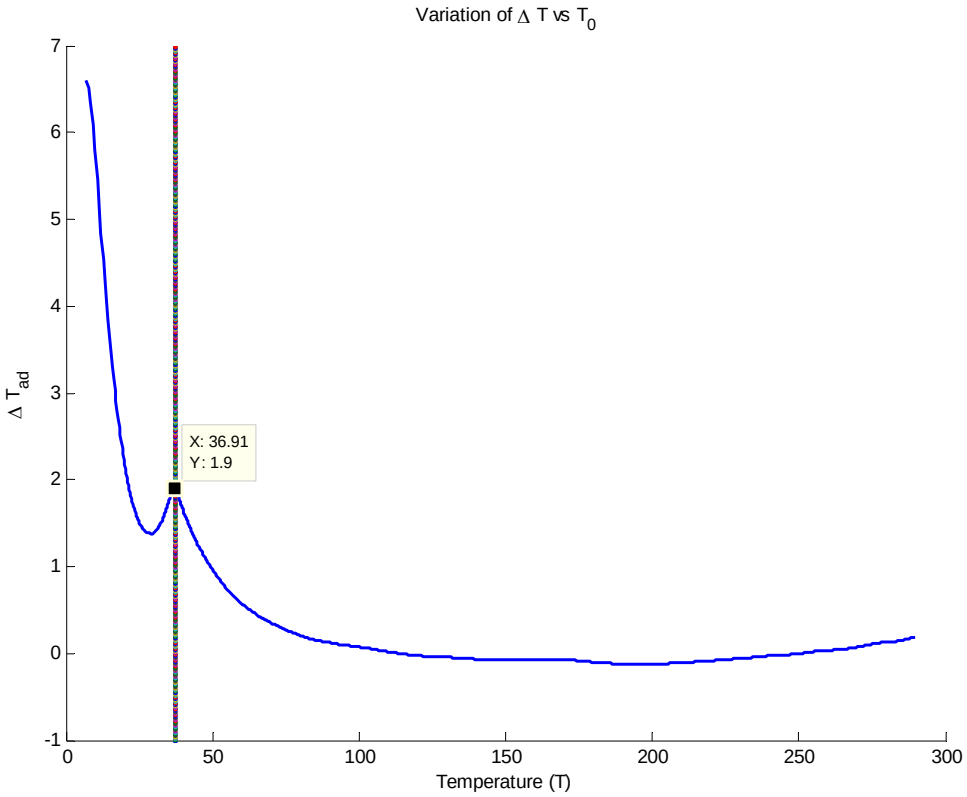
More Closely



$\Delta S_m(T, \Delta H)$



$\Delta T_{ad}(T, \Delta H)$



Quantitative results

$\Delta S_{m,max} = 1.3213$

$T = 37.0000$

$\Delta T_{ad,max} = 1.9092$

$T = 36.9081$

Matlab code

```
clear all;
close all;
clc;

importdata('TbRu2Ge2_zero.dat');
F_0 = ans.data(:,6);
T_0 = ans.data(:,8);
C_0 = ans.data(:,10);

% To make the data increasing
F_0 = F_0(length(F_0):-1:1);
T_0 = T_0(length(T_0):-1:1);
C_0 = C_0(length(C_0):-1:1);

S0_0 = C_0(1) / T_0(1)/2;

% Temperature range for interpolated data
T_I  = 2.7:0.1:296;

% Get interpolated C
C_0_I = interp1 (T_0,C_0,T_I);

% To plot the values of C vs T
figure
plot(T_I,C_0_I,'LineWidth',1.5);
hold all;

I_0 = zeros(1,length(T_I));
CbyT_0 = C_0_I./T_I;
for n=2:length(T_I)
    I_0(n) = trapz(T_I(1:n),CbyT_0(1:n));
end

% another data for magnetic...
importdata('TbRu2Ge2_50kOe.dat');
F_50 = ans.data(:,6);
T_50 = ans.data(:,8);
C_50 = ans.data(:,10);

% To make the data increasing
F_50 = F_50(length(F_50):-1:1);
T_50 = T_50(length(T_50):-1:1);
```

```
C_50 = C_50(length(C_50):-1:1);

%S0_50 = C_50(1)/T_50(1)/2;
S0_50 = S0_0;

% Get interpolated C
C_50_I = interp1 (T_50,C_50,T_I);

% To plot the values of C vs T
plot(T_I,C_50_I,'LineWidth',1.5);
xlabel ('Temperature (K)')
ylabel ('Specific heat capacity')
title ('Specific heat capacity vs Temperature at
various values of magnetic field ')
legend('0 KOestred','50 KOestred');

I_50 = zeros(1,length(T_I));
CbyT_50 = C_50_I./T_I;
for n=2:length(T_I)
    I_50(n) = trapz(T_I(1:n),CbyT_50(1:n));
end

% New figure to plot the values of entropy
figure;
% To plot the values of S_0
S_0 = I_0 + S0_0;
plot(T_I,S_0,'LineWidth',1.5);
hold all;

S_50 = I_50 + S0_50;
plot(T_I,S_50,'LineWidth',1.5);

% Now find deltaSM
SM_0_50 = S_0-S_50;
plot(T_I,SM_0_50,'LineWidth',1.5);
hold on;

% Highlight maximum
[M_S_0_50,I] = max(SM_0_50);
T_I(I);

plot(T_I(I),-10:0.01:150)
```

```
xlabel ('Temperature (K)')
ylabel ('Entropy (S)')
title ('Entropy vs Temperature at various values of
magnetic field')
legend('0 KOestred','50 KOestred','\Delta S_M');

xlabel ('Temperature (K)')
ylabel ('Entropy (S)')

% To plot delta SM_0_50 in a separate plot
figure
hold on;
xlabel ('Temperature (K)')
ylabel ('Entropy (\Delta S)')
title ('\Delta S_M vs Temperature')
plot(T_I,SM_0_50,'LineWidth',1.5);
hold on;

% Highlight maximum
disp ('The maximum of Delta S_M');
[M_S_0_50,I] = max(SM_0_50)
T_I(I)

plot(T_I(I),0:0.01:1.5)

% TO FIND DELTA TM FROM DELTA SM DATA
% first we have to interpolate the data
figure
S_I = 1:0.1:180;
T_0_I = interp1 (S_0,T_I,S_I);
T_50_I = interp1 (S_50,T_I,S_I);

T_0_50_I = T_50_I - T_0_I;

plot(S_I,T_0_50_I,'LineWidth',1.5);
hold all;

plot(S_I,T_0_I)
plot(S_I,T_50_I)

legend('T_5_0_,_I','T_0_I','T_5_0_I');
```

```
% Highlight maximum
disp ('The overall maximum of Delta T');
[M_T_0_50 I] = max(T_0_50_I)
S_I(I)
plot(S_I(I),-1:0.01:7)
hold on;

xlim([0 50]);

title ('Variation of \delta T with S')
xlabel('Entropy (S)');
ylabel('Difference in Temperature \delta T')

figure
hold on;
title('Variation of \Delta T_a_d vs T_0');
xlabel ('Temperature (T)');
ylabel('\Delta T_a_d');

plot(T_0_I,T_0_50_I,'LineWidth',1.5);

% plot max
disp ('The relevant maximum of Delta T vs T_0');
[M_T_0_50 I] = max(T_0_50_I(78:end))
I=I+78;
T_0_I(I)

% To highlight maximum
plot(T_0_I(I),-1:0.01:7)
```

In this code data is first interpolated because the specific heats at the two values of magnetic fields at not at same temperature, so in principal they can't be directly negated to obtain $\Delta S_m(T, \Delta H)$. To overcome this problem both the specific heats are interpolated to get y values at common temperature values.

Program Output

```
The maximum of Delta S_M

M_S_0_50 =

    1.3213

I =

   344

ans =

   37.0000


The overall maximum of Delta T

M_T_0_50 =

    6.5946

I =

     1

ans =

     1


The relevant maximum of Delta T vs T_0

M_T_0_50 =

    1.9092

I =

    51

ans =

   36.9081
```

BIBLIOGRAPHY

Research papers

Magnetocaloric effect and magnetic refrigeration – Vitalij K pecharsky, Karl A. Gshneidner Jr.

Journal of Magnetism and Magnetic Materials 200(1999) 44-56

On the low-temperature properties of TmRu2Si2 – Lukasz Gondek, Dariusz Kaczorowski, Andrzej Szytula

Solid State Communications (www.elsevier.com/locate/ssc)

Books

1. Physical modelling in Matlab – Allen B Downey
2. Engineering and Scientific Computations Using MATLAB - Sergey E. Lyshevski
3. Essential MATLAB for Engineers and Scientists - Brian D. Hahn & Daniel T. Valentine
4. Computational Statistics Handbook with MATLAB - Martinez & Martinez
5. Fundamentals of Electromagnetics with Matlab - Lonngren & Savov
6. Specific heats at low temperatures – E S R Gopal

Web courses

OCW - 6-00 Introduction to computer science and programming - Fall-2008
Prof. Eric Grimson
Prof. John Guttag

Web links

1. http://en.wikipedia.org/wiki/Conceptual_model
2. http://en.wikipedia.org/wiki/Scientific_modelling
3. http://en.wikipedia.org/wiki/Mathematical_model
4. http://en.wikipedia.org/wiki/Simulation
5. http://en.wikipedia.org/wiki/Dynamical_system
6. http://en.wikipedia.org/wiki/Taylor_series
7. http://en.wikipedia.org/wiki/Chaos_theory
8. http://en.wikipedia.org/wiki/Stochastic_process
9. http://en.wikipedia.org/wiki/Statistics
10. http://en.wikipedia.org/wiki/Pseudorandom_number_generator
11. http://en.wikipedia.org/wiki/Random_walk
12. http://en.wikipedia.org/wiki/Monte_Carlo_method
13. http://home.freeuk.net/alunw/mandelbrotroom.html