Problem 1:

Create different two dimensional phase portraits of a time series of the Henon map (Example 3.2) using delay times tau = 1, 2, 3, ... (here, the sampling time is 1). Which picture gives the clearest information? Rewrite the map in delay coordinates with unit delay.

Solution:

The Henon model or Henon map is interesting in that it consists of two variables.
Both chaotic and periodic attractors may be found for the Henon model depending on the choice of the control parameters a and b.

$$X_{n+1} = 1 - aX_n^{\wedge} 2 + Y_n$$

$$Y_{n+1} = b X_n$$

The first plot shows Henon map for the initial conditions (a = 0.9 and b = 0.3).. henon_1 shows the plot.

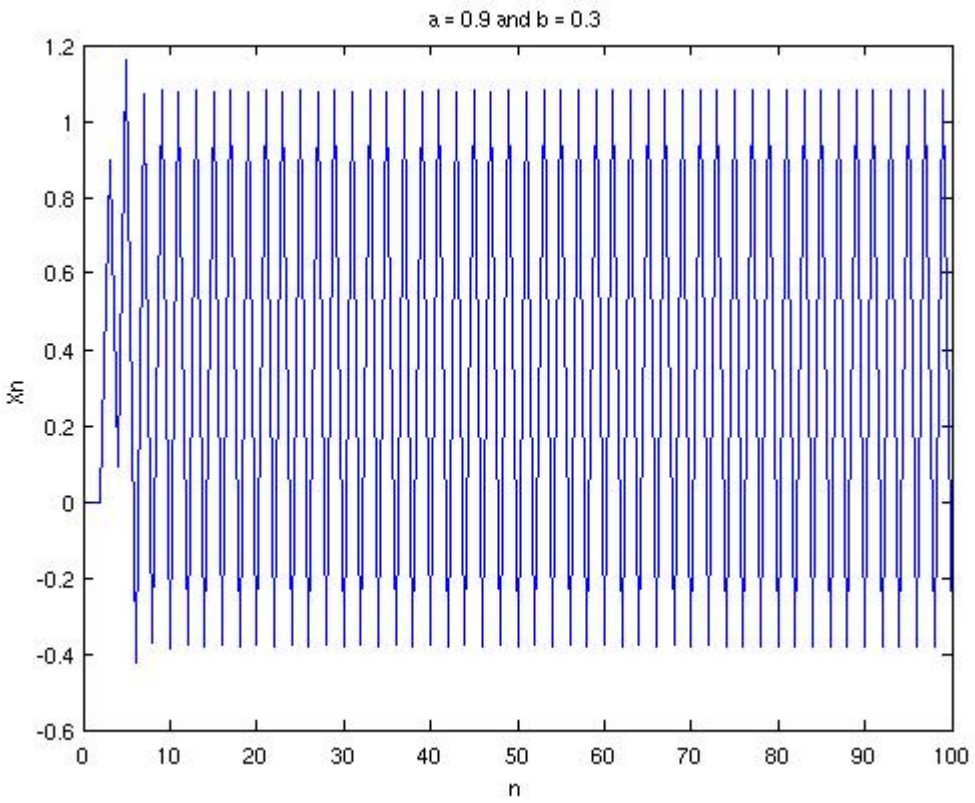

Fig 1.1: Henon at a = 0.9 and b = 0.3

The plot 2 (henon_2) shows the Henon attractor for a = 1.4 and b = 1.3).

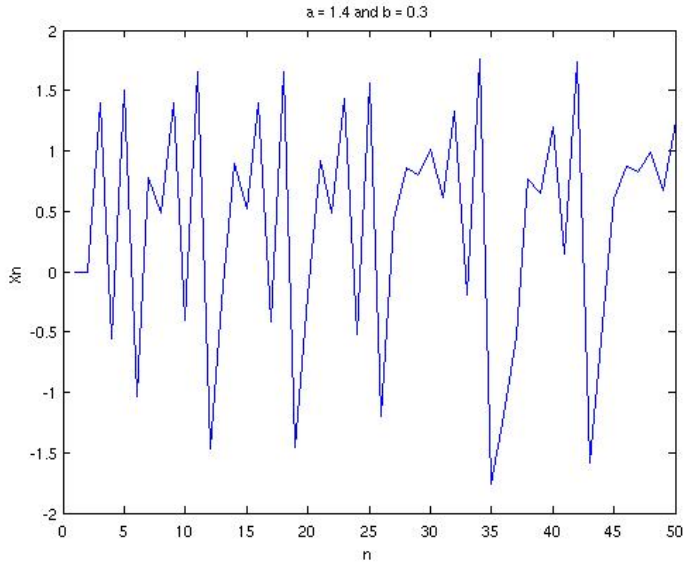

Fig 1.2: Henon at a = 1.4 and b = 1.3

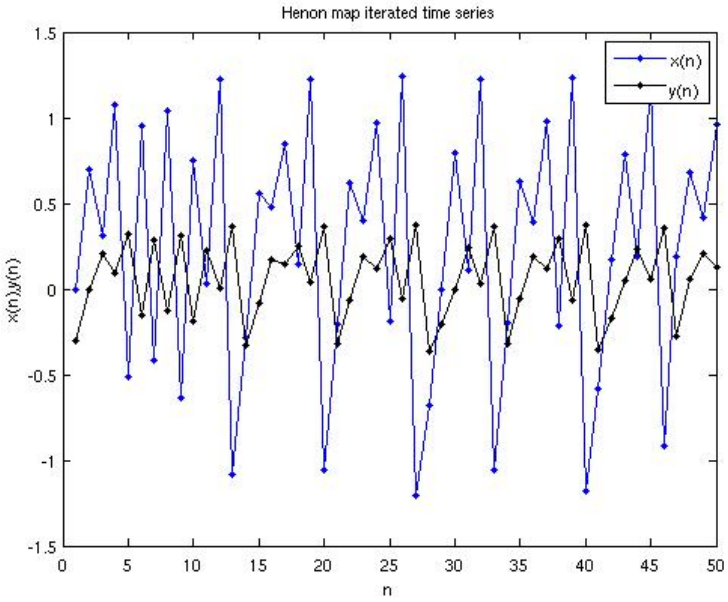

Fig 1.3: Henon map iterated time series

The plot 4 (henon_4) shows the whole attractor. Using the same initial conditions as a = 1.4 and b = 0.3, it can be observed that the strange attractor has a shape of the letter C

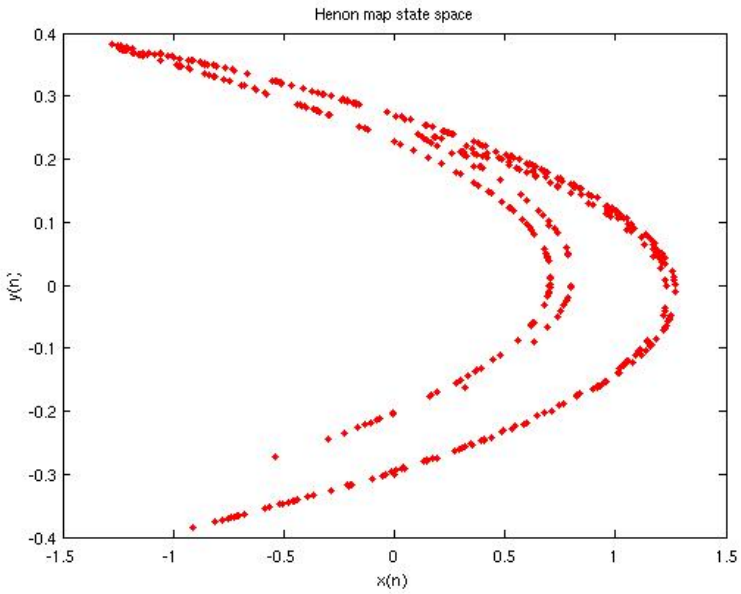

Fig 1.4: Henon map state space

MATLAB CODE:

```
% Henon map
% Initial conditions
N = 500;
M = 50;

% Henon map parameters
% henon map is usually very sensitive to the intital
%conditions
a = 1.4;
b = 0.3;

% Initial conditions
x(1) = 0;
y(1) = -0.3;

% Iterate the map
for n = 1:(N-1)
   x(n+1) = 1-a*x(n)^2+y(n);
   y(n+1) = b*x(n);
end

% Plot iterated time series of x and y
figure;
plot(1:M, x(1:M), 'b.-', 1:M, y(1:M), 'k.-');
title('Henon map iterated time series');
legend('x(n)','y(n)');
xlabel('n');
ylabel('x(n),y(n)');
```

```
% Plot of state space
figure;
plot(x, y, 'r.');
title('Henon map state space');
xlabel('x(n)');
ylabel('y(n)');
```

Problem 2:

Create a scalar time series of flow data numerically by integration of the Lorenz system described by the set of differential equations:
dx/dt = sigma*(y-x)
dy/dt = r*x - y - y*z
dz/dt = -b*z +x*y
with parameters sigma = 10, r = 28, b = 8/3. Use the Runge-Kutta(**) numerical integration technique to solve these set of equations to obtain one scalar observable (e.g., x).
Plot the reconstructed attractor in 2-D delay coordinates with different time lags. Convince yourself (by the plots) that the reconstruction is best when the lag is about one-quarter of the mean cycle time (note that if a time series has a strongly periodic component, an optimal time delay is approximately a quarter of the cycle time). Compute the autocorrelation function(as required in problem 4 below) as a function of tau for this time series and confirm this impression.

Solution:

The Lorenz attractor is plotted varying the number of points first. But the actual Lorentz time series is shown in the figure Lorenz_time_series. As can be seen from the plot, we cannot say that Lorentz is periodic.

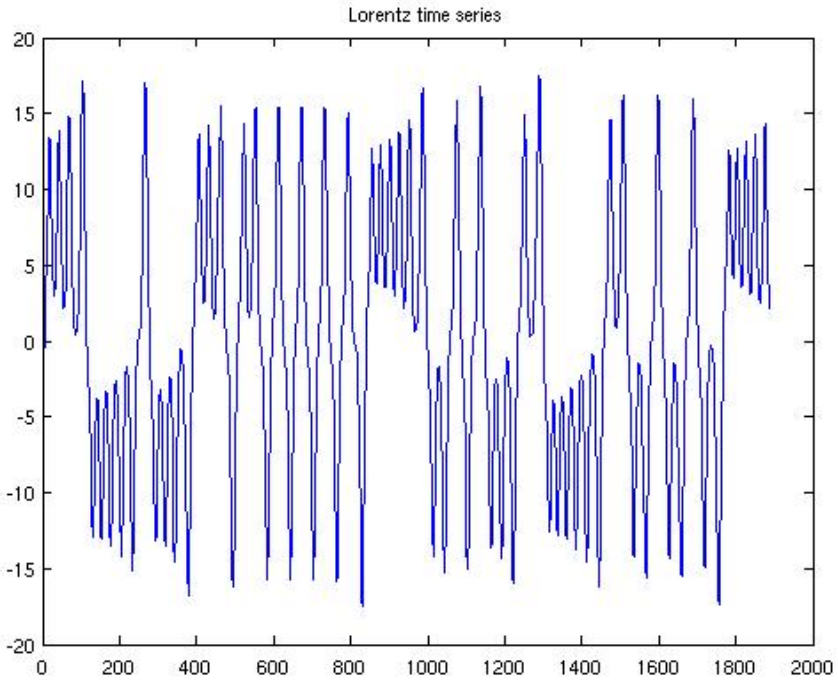

Fig 2.1: Lorenz time series

The plots lorentz_1 and lorentz_2 show the lorentz time series for varying number of data points and in different axes. Clearly from the plots N = 10,000 has given out a better attractor than N = 500 case.

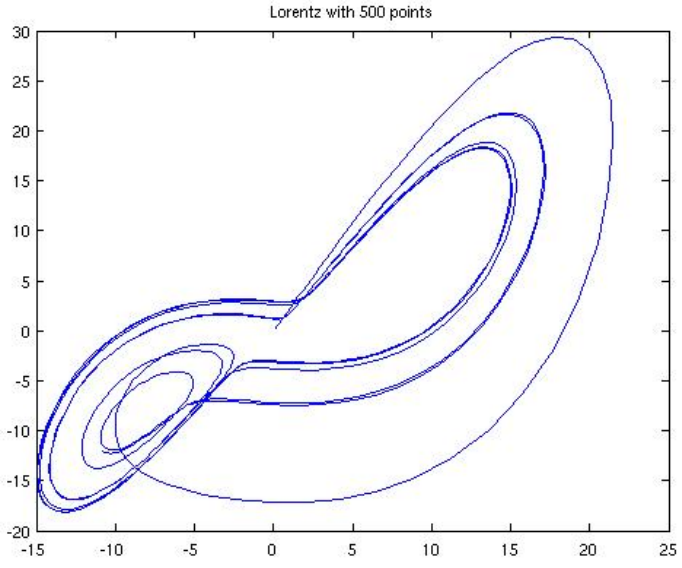

Fig 2.2: Lorenz attractor for N = 500 points

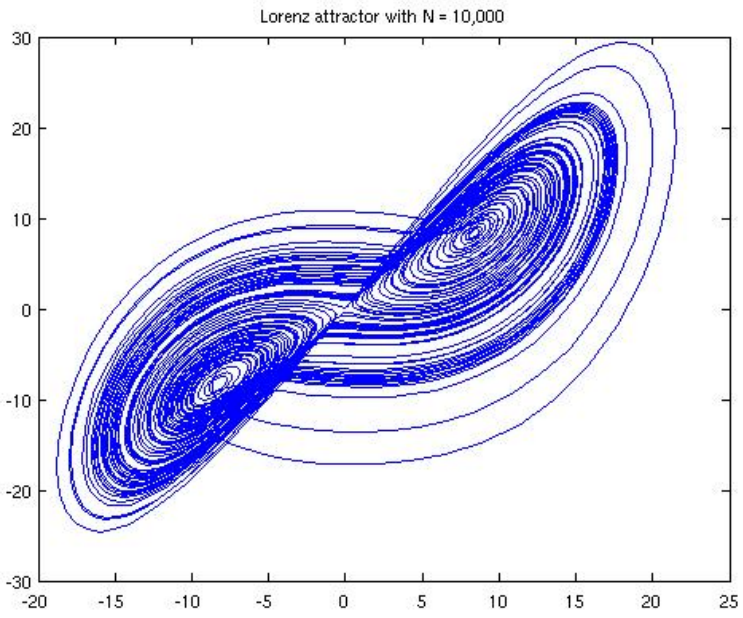

Fig 2.3: Lorenz attractor for N = 10,000 points

The Lorentz attractor that is shown above is the actual attractor. With variation in the value of tau, the attractor also varies. This can be shown in the plots below.

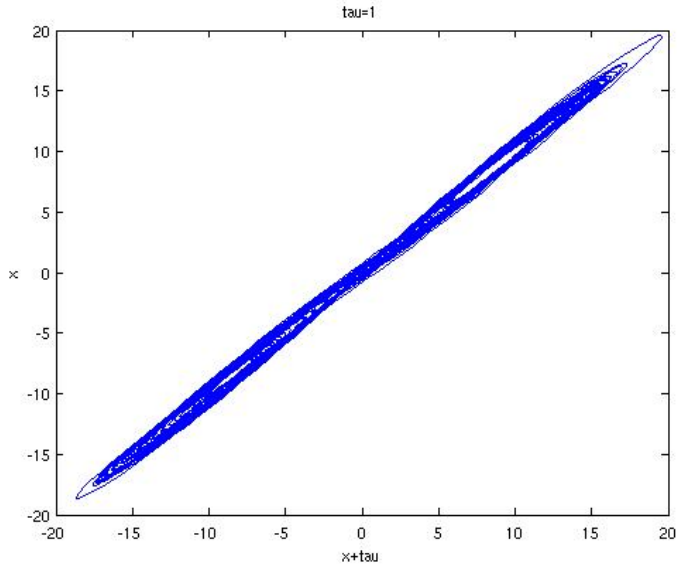

Fig 2.3: Attractor when tau = 1 (almost at 45 degrees)

This is the attractor when the value of time delay that is chosen in 1. As can be seen, the attractor is not spread out at all. The attractor is closed up and aligned at 45 degrees. With increase in tau value, the attractor starts spreading. It can be observed here:

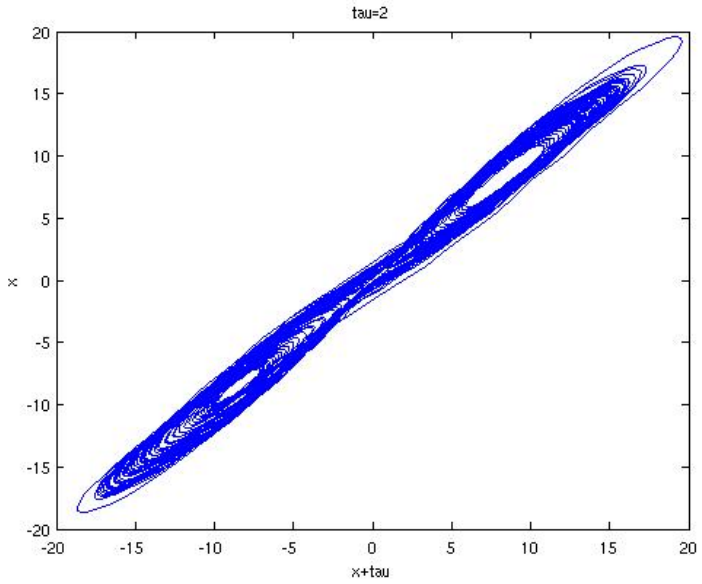

Fig 2.4: Attractor at tau = 2

With only slight increase in the tau value the attractor appers to be spread out. The spread can be observed clearly as the value of tau keeps increasing. The plots below show the attractor with increasing tau value.

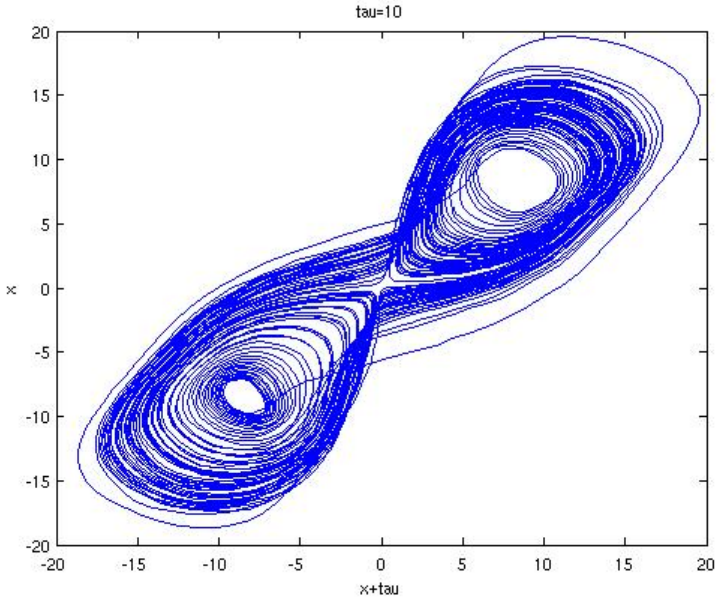

Fig 2.5: Attractor at tau = 10

The attractor completely opens up when the tau value is 15, which is approximately 1/4th the time period of Lorenz time series.

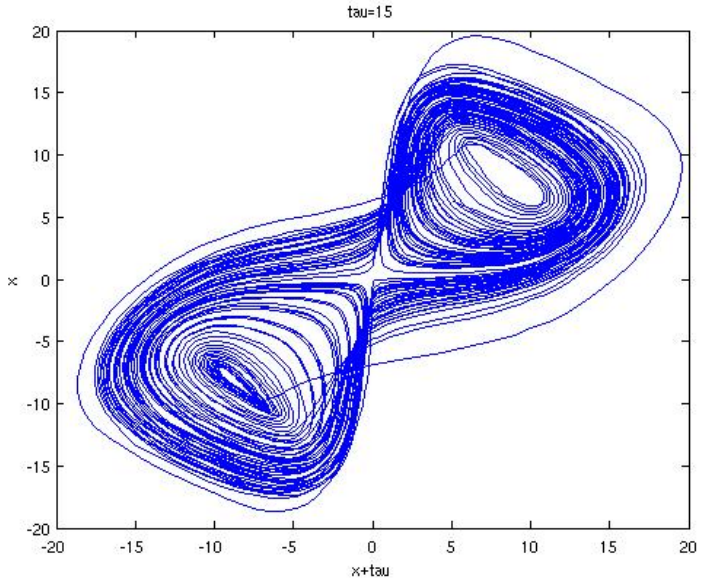

Fig 2.6: Attractor at tau = 15

As I increase the value of tau, the attractor becomes more difficult to discern and becomes unrecognizable.

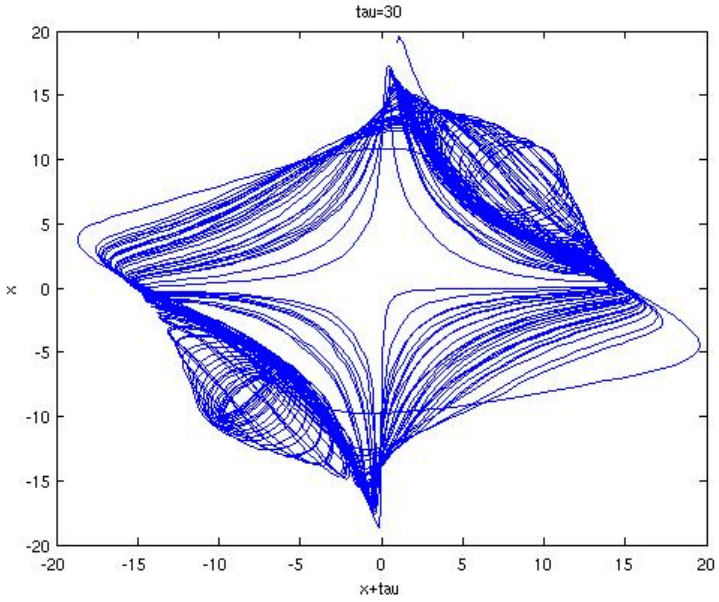

Fig 2.7 : Attractor at tau = 30 (more than optimum)

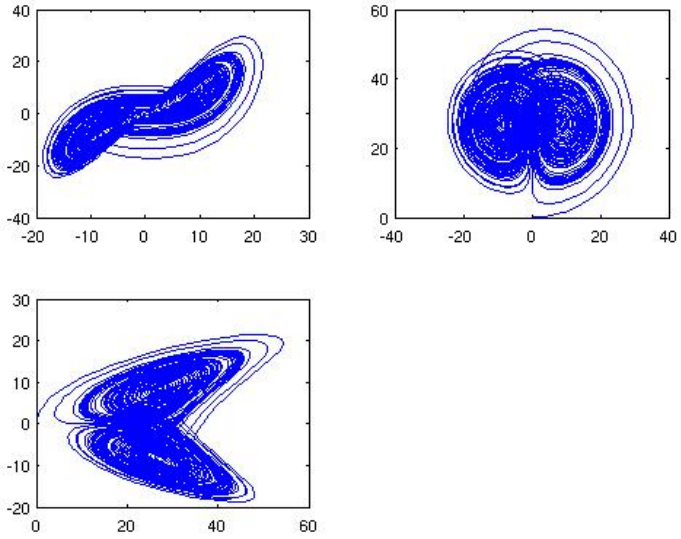

Fig 2.8: Attractor at different views

From the attractors observed, the tau value between 10 and 12 appears appropriate.

MATLAB CODE:

```
function [x,y,z]=lorentz(n,level,s,r,b,x0,y0,z0,h)
%
%    dx/dt=s*(y-x)
%    dy/dt=r*x-y-xz
%    dz/dt=x*y-b*z;
%
% x, y, and z are the simulated time series.
% n is the number of the simulated points.
% level is the noise standard deviation divided by the standard
deviation of the
%   noise-free time series. We assume Gaussian noise with zero mean.
% s, r, b, and s are the parameters
% x0 is the initial value for x.
% y0 is the initial value for y.
% z0 is the initial value for z.
% h is the step size.
%

if nargin<1 | isempty(n)==1
    n=10000;
else
    % n must be scalar
    if sum(size(n))>2
        error('n must be scalar.');
    end
    % n must be positive
    if n<0
        error('n must be positive.');
    end
```

```
    % n must be an integer
    if round(n)-n~=0
        error('n must be an integer.');
    end
end

if nargin<2 | isempty(level)==1
    level=0;
else
    % level must be scalar
    if sum(size(level))>2
        error('level must be scalar.');
    end
    % level must be positive
    if level<0
        error('level must be positive.');
    end
end

if nargin<3 | isempty(s)==1
    s=10;
else
    % s must be scalar
    if sum(size(s))>2
        error('s must be scalar.');
    end
end

if nargin<4 | isempty(r)==1
    r= 28;
else
    % r must be scalar
    if sum(size(r))>2
        error('r must be scalar.');
    end
end

if nargin<5 | isempty(b)==1
    b=2.6;
else
    % b must be scalar
    if sum(size(b))>2
        error('b must be scalar.');
    end
end

if nargin<6 | isempty(x0)==1
    x0=0.1;
else
    % x0 must be scalar
    if sum(size(x0))>2
        error('x0 must be scalar.');
    end
end

if nargin<7 | isempty(y0)==1
    y0=0.1;
```

```
else
    % y0 must be scalar
    if max(size(y0))>2
        error('y0 must be scalar.');
    end
end

if nargin<8 | isempty(z0)==1
    z0=0.1;
else
    % z0 must be scalar
    if max(size(z0))>2
        error('z0 must be scalar.');
    end
end

if nargin<9 | isempty(h)==1
   h=0.01;
else
    % h must be scalar
    if max(size(h))>2
        error('h must be scalar.');
    end
    % h must be positive
    if h<0
        error('h must be positive.');
    end
end

% Initialize
y(1,:)=[x0 y0 z0];

% Simulate
for i=2:n
ydot(1)=s*(y(i-1,2)-y(i-1,1));
ydot(2)=r*y(i-1,1)-y(i-1,2)-y(i-1,1)*y(i-1,3);
ydot(3)=y(i-1,1)*y(i-1,2)-b*y(i-1,3);
y(i,:)=y(i-1,:)+h*ydot;
end

% Separate the solutions
x=y(:,1);
z=y(:,3);
y=y(:,2);

% Add normal white noise
x=x+randn(n,1)*level*std(x);
y=y+randn(n,1)*level*std(y);
z=z+randn(n,1)*level*std(z);
subplot(2,2,1);
plot(x,y);
subplot(2,2,2);
plot(y, z);
subplot(2,2,3);
plot(z, x);
```

Problem 3:

The phase portrait is simplified by plotting poincare map. Points in the poincare map for the chaotic system never reappear at the same location.

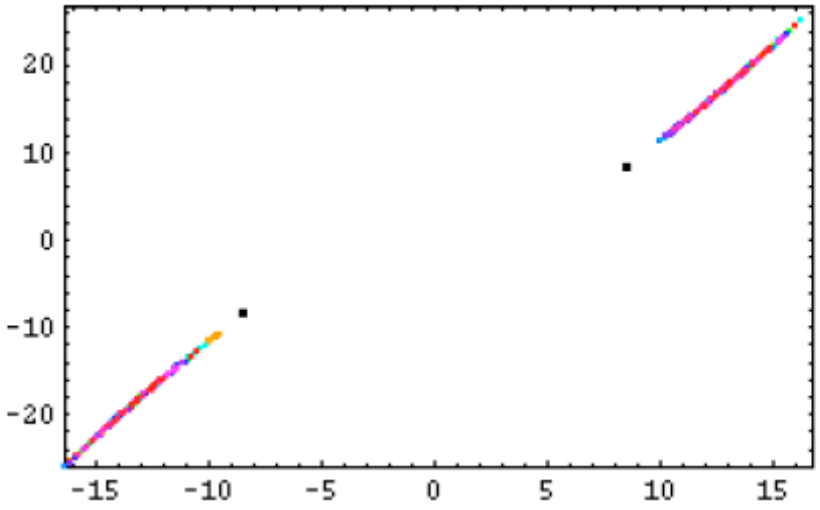

Fig 3.1 : Pincare section for LorenZ Z = 21

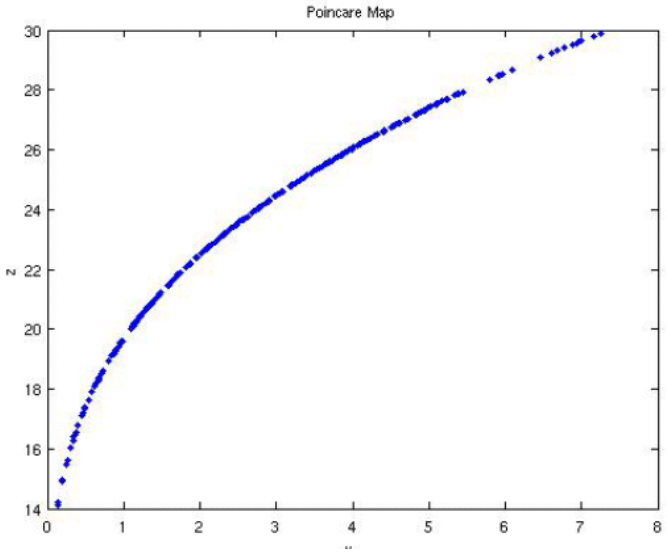

Fig 3. 2: Poincare map for X = 0

MATLAB CODE:

```
function lormap = poincare(plane_d, trajec, derivative, derivative_wrt)
%
% Computes a Poincare map of a trajectory based on an arbitrary plane
%
% arguments:
%  plane_d: the three points that define the plane, i.e.
%  trajec: trajectory (3 dimensions)
%  derivative: either 'positive' or 'negative'
%  derivative_wrt: derivative with respect to what variable (1=x, 2=y, 3=z)
%
% return:
%  map: a 3xm matrix containing intersection points, i.e. the Poincare map
%

% get the length of the time series
%
l = length(xyz(:,1));

% initialize the poincare map
%
map=[];

% loop over time series
%
for i=1:l-1

   % determine the sign of the derivative between these two points
   %
   delta = xyz(i,derivative_wrt)-xyz(i+1,derivative_wrt);

   % only continue if sign agrees with 'derivative' argument
   %
   if (((delta < 0) & (derivative == 'negative')) |...
      ((delta > 0) & (derivative == 'positive')))

      % determine whether the line formed by the two points intersects
      % the plane defined by the user
      %
      line = [xyz(i,:);xyz(i+1,:)];

      den = [[1 1 1 0];
            [plane(1,1),plane(2,1),plane(3,1),line(2,1)-line(1,1)];
            [plane(1,2),plane(2,2),plane(3,2),line(2,2)-line(1,2)];
            [plane(1,3),plane(2,3),plane(3,3),line(2,3)-line(1,3)]];
```

```
        num = [[1 1 1 1];
              [plane(1,1),plane(2,1),plane(3,1),line(1,1)];
              [plane(1,2),plane(2,2),plane(3,2),line(1,2)];
              [plane(1,3),plane(2,3),plane(3,3),line(1,3)]];

        t = -det(num)/det(den);

        x = line(1,1) + (line(2,1)-line(1,1))*t;
        y = line(1,2) + (line(2,2)-line(1,2))*t;
        z = line(1,3) + (line(2,3)-line(1,3))*t;

        % the point at which the line intersects the user defined plane
        %
        point = [x,y,z];

        % determine whether or not this point of intersection lies between
        % the current two points
        %
        if (((derivative == 'negative') &...
             (point(derivative_wrt)-xyz(i,derivative_wrt) > 0) &...
             (point(derivative_wrt)-xyz(i+1,derivative_wrt) < 0)) |...
            ((derivative == 'positive') &...
             (point(derivative_wrt)-xyz(i,derivative_wrt) < 0) &...
             (point(derivative_wrt)-xyz(i+1,derivative_wrt) > 0)))

            % if so, add this point to the Poincare map
            %
            map = [map;[x,y,z]];
        end
    end
end
```

Problem 4:

Auto Mutual Information:

The mutual information between xi generated from system X and yj generated from system Y is the amount of information that xi provides about yj.
The mutual information is a measure of dynamical coupling or information transmission between X and Y. The AMI estimates the degree to which the x(t + tau) can be estimated from x(t) or the mean predictability of future points in the same time series from past points. AMI is considered the non-linear version of auto correlation function. Mutual

Information is preferred over correlation as MI is a function of both linear and non-linear dependencies between two variables.
The plot for the auto correlation for Lorentz time series (this is plotted using our IFc's) is shown in fig 4.1

IFC code for finding correlation for Lorenz:

```
// isisp include files
//
#include <Correlation.h>
#include <VectorFloat.h>
#include <Generator.h>

int main() {

// declare the Correlation object
//
Correlation acor;

// declare the input and output vectors
//
 VectorFloat input, dummy;
 VectorFloat output;
 Generator gen1;
 gen1.set(Generator::LORENTZ);
 gen1.setSampleFrequency((float)100);
 gen1.setGenSampleFrequency((float)100);
 gen1.setSignalDuration(10);
 gen1.setFrameDuration(10);
 gen1.compute(input, dummy);
 input.debug(Linput >>);

// set the algorithm, implementation, normalization, and order
//
 acor.setAlgorithm(Correlation::AUTO);
 acor.setImplementation(Correlation::FACTORED);
 acor.setNormalization(Correlation::UNIT_ENERGY);
 acor.setOrder(100);

 // compute the autocorrelation output
 //
 acor.compute(output, input);
 output.debug(Loutput>>);
```

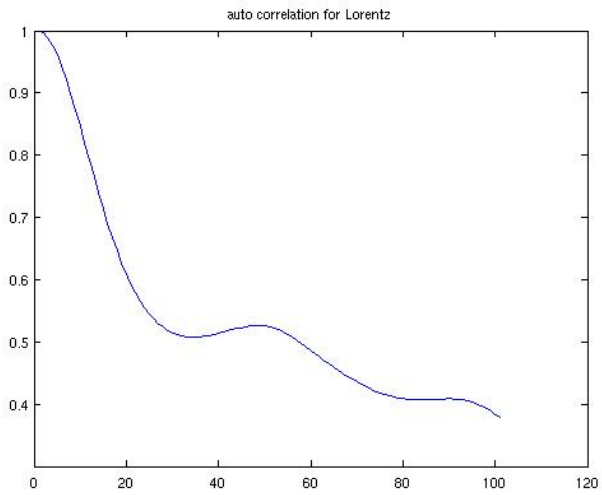

Fig 4.1: Auto correlation for Lorentz time series

Observing for the minimum value we find that the minimum value is obtained at tau = 30 which is not correct.