IE 597 Homework 2

Fall 2020

Haedong Kim

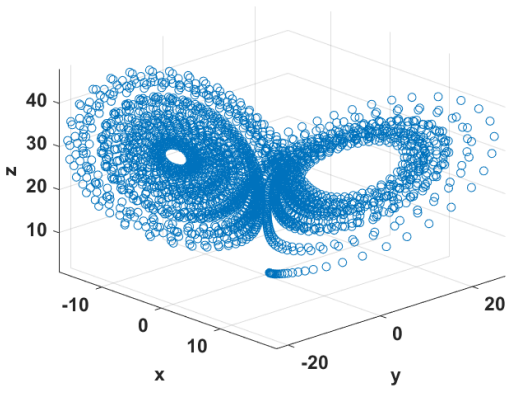
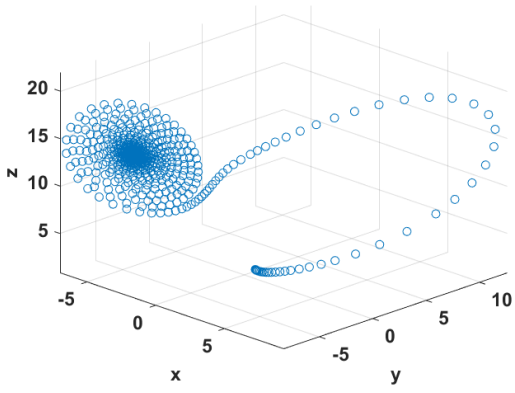
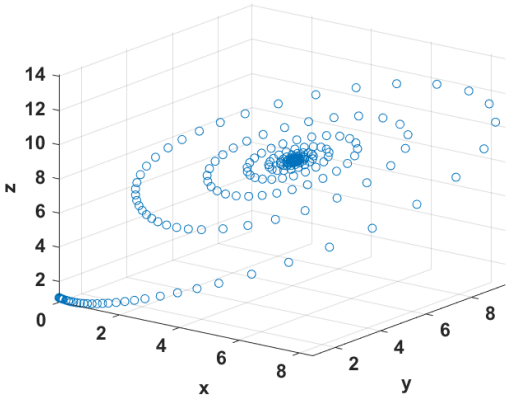
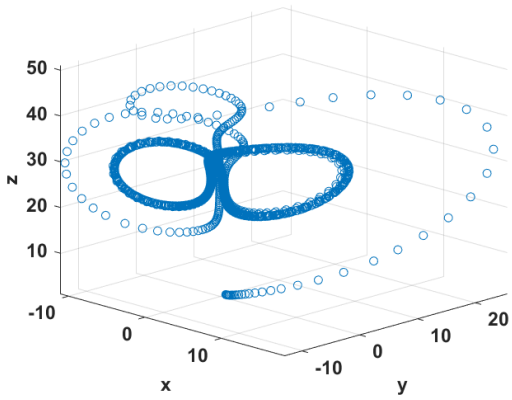
Problem 1. Lorenz and Rossler systems

1.1. Lorenz attractors with different parameters

The Lorenz system is parameterized as below with 3 parameters (p_1, p_2, p_3)

$$\frac{dx}{dt} = p_1(y - x)$$
$$\frac{dy}{dt} = x(p_2 - z) - y$$
$$\frac{dz}{dt} = xy - p_3 z$$

Table 1. Lorenz attractors with different sets of parameters

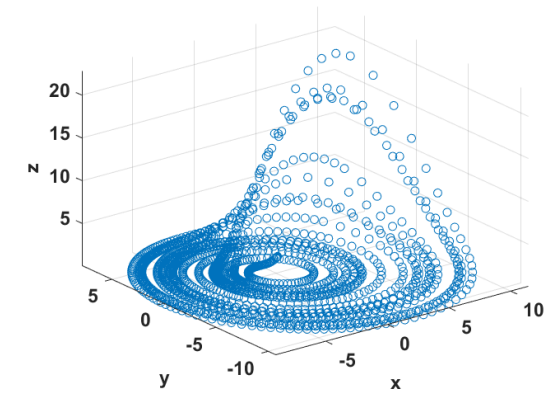
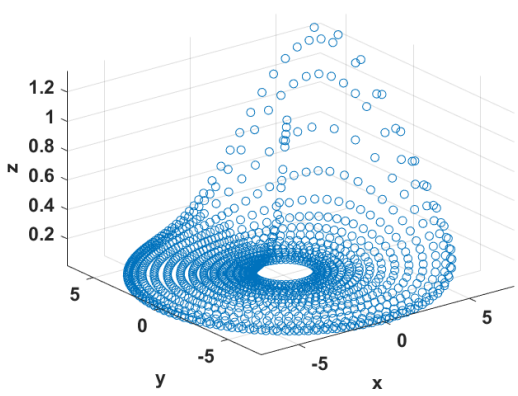
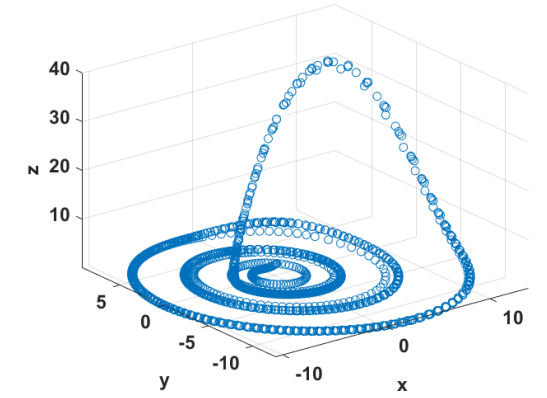
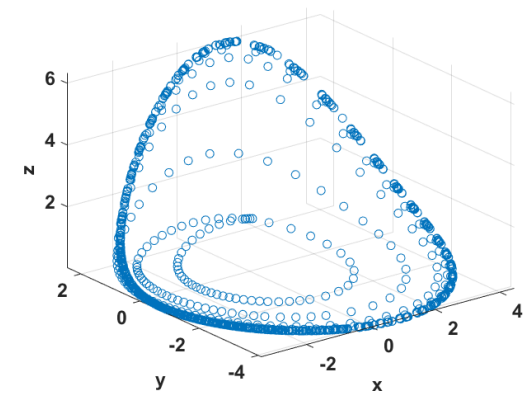
$(10, 28, 8/3)$ 	$(5, 14, 8/5)$ 
$(10, 10, 8/3)$ 	$(10, 10, 8/3)$ 

1.2. Rossler attractors with different parameters

The Rossler system is parameterized as below with 3 parameters (p_1, p_2, p_3)

$$\frac{dx}{dt} = -y - z$$
$$\frac{dy}{dt} = x + p_1 y$$
$$\frac{dz}{dt} = p_2 + z(x - p_3)$$

Table 2. Rossler attractors with different sets of parameters

$(0.2, 0.2, 5.7)$ 	$(0.05, 0.2, 5.7)$ 
$(0.2, 0.05, 5.7)$ 	$(0.2, 0.2, 1)$ 

Problem 2. Bifurcation diagram for logistic map

$x(0) = 0.1$

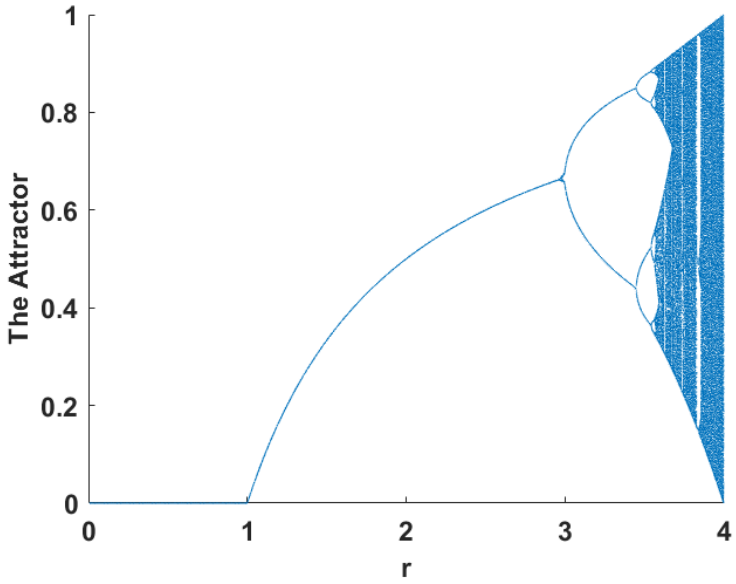


Figure 1. Bifurcation diagram for logistic map

Problem 3.

Appendix [5] (mutual_infor.m) and [6] (FNN.m) includes mutual information and false nearest neighbor (FNN) tests respectively.

Problem 4.

X-dimension of the Rossler system with parameters $(0.2, 0.2, 5.7)$ is taken as input signal. The result of mutual information test is shown in Figure 2.

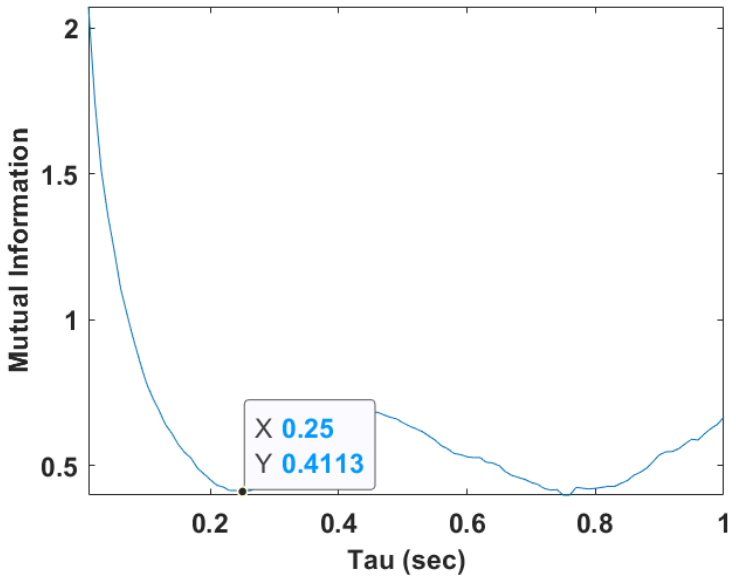


Figure 2. Mutual information over time lags

The first local minum 0.25 sec is selected as the time lag and FNN test is performed with this tau. Tolerance for the FNN criterion was 5 and 1st nearest neighbors were used. The result is given in Figure 3.

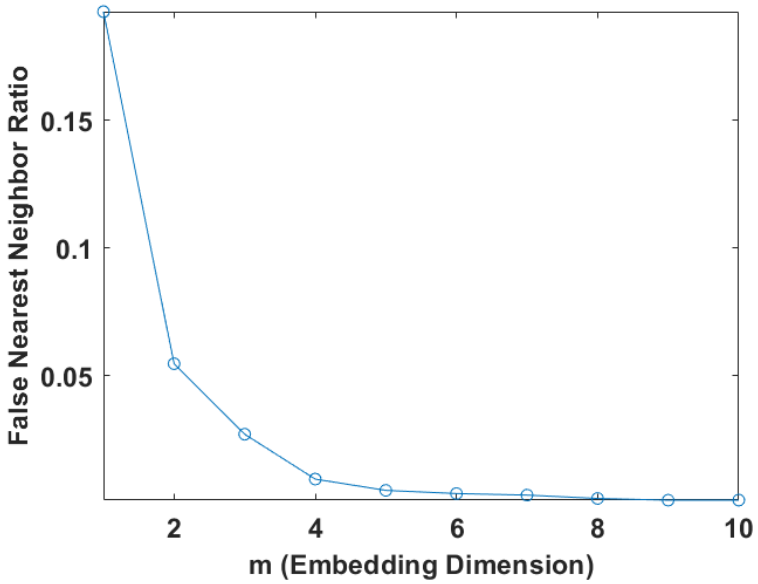


Figure 3. FNN with 0.25-time lag

Appendix - MATLAB Code

[1] HW2_haedong.m (main script)

```
%% Problem 1 - Lorenz
clc
close all
clear variables

lparam0 = [10, 28, 8/3];
linit = [0, 1, 1.05];
[~, s1] = Lorenz(lparam0, linit, 50);

scatter3(s1(:,1), s1(:,2), s1(:,3))
xlabel('x'); ylabel('y'); zlabel('z')
set(gca,'CameraPosition', [205.8004 -261.1243  213.1089])
axis tight

%% Problem 1 - Roslle r
clc
close all
clear variables

rparam0 = [0.2, 0.2, 5.7];
rinit = [1, 1, 1];
[~, s2] = Roslle r(rparam0, rinit, 100);

figure(2)
scatter3(s2(:,1), s2(:,2), s2(:,3))
xlabel('x'); ylabel('y'); zlabel('z')
```

```
axis tight

%% Problem 2
clc
close all
clear variables

r = 0.01:0.001:4;
xpts = [];
ypts = [];
for i=1:length(r)
    fprintf('###ITER: %i/%i \n', i, length(r))
    lims = log_growth_lim(0.1, r(i), 50);
    x = r(i)*ones(1, length(lims));
    xpts = [xpts, x];
    ypts = [ypts, lims];
end

scatter(xpts, ypts, 0.15)
xlabel('r'); ylabel('the Attractor')

%% Problem 4 - Mutual Information
clc
close all
clear variables

rparam0 = [0.2, 0.2, 5.7];
rinit = [1, 1, 1];
[~, s2] = Roslle(rparam0, rinit, 100);

x = s2(:,1);

% tau is multiplier of time-step of ODE solver; approximate default is 0.01
tau = 1:100;
t = tau * 0.01;

mis = zeros(1, length(tau));
for i=1:length(tau)
    mis(i) = mutual_info(x, tau(i), 20);
end

plot(t, mis)
axis tight
xlabel('Tau (sec)')
ylabel('Mutual Information')

%% Problem 4 - FNN
clc
close all
clear variables

rparam0 = [0.2, 0.2, 5.7];
rinit = [1, 1, 1];
[~, s2] = Roslle(rparam0, rinit, 100);

x = s2(:,1);

% tau is multiplier of time-step of ODE solver; approximate default is 0.01
tau = 1;

% size of dimension
```

```
dim = 1:1:10;

% rth nearest neighbor
r = 1;

% FNN-criterion tolerance
tol = 5;

fnn_ratios = FNN(x, tau, dim, r, tol);
plot(dim, fnn_ratios, '-o')
axis tight
xlabel('m (Embedding Dimension)')
ylabel('False Nearest Neighbor Ratio')
```

[2] Lorenz.m

```
function [t, states] = Lorenz(param, init, sim_time)
    init_vals = init;
    tspan = [0, sim_time];
    [t, states] = ode45(@(t, states)compute_rates(t, states, param), tspan, init_vals);
end

function rates = compute_rates(t, states, param)
    % {states(1): x, states(2): y, states(3): z}
    states_size = size(states);
    states_ncol = states_size(2);
    states_nrow = states_size(1);
    if (states_ncol == 1)
        states = states';
    else
        rates = zeros(states_nrow, states_ncol);
    end

    rates(:,1) = param(1).*(states(:,2)-states(:,1));
    rates(:,2) = states(:,1).*(param(2)-states(:,3))-states(:,2);
    rates(:,3) = states(:,1).*states(:,2)-param(3).*states(:,3);

    rates = rates';
end
```

[3] Rosller.m

```
function [t, states] = Rosller(param, init, sim_time)
    init_vals = init;
    tspan = [0, sim_time];
    [t, states] = ode45(@(t, states)compute_rates(t, states, param), tspan, init_vals);
end

function rates = compute_rates(t, states, param)
    % {states(1): x, states(2): y, states(3): z}
    states_size = size(states);
    states_ncol = states_size(2);
```

```
    states_nrow = states_size(1);
    if (states_ncol == 1)
        states = states';
    else
        rates = zeros(states_nrow, states_ncol);
    end

    rates(:,1) = -states(:,2)-states(:,3);
    rates(:,2) = states(:,1)+param(1).*states(:,2);
    rates(:,3) = param(2)+states(:,3).*(states(:,1)-param(3));

    rates = rates';
end
```

[4] log_growth_lim.m

```
function [lims, e] = log_growth_lim(x0, r, step_size)
    e = []; % expectations of batches
    d = []; % differences of expections of batches
    cnt = 1; % counter of while loop
    tol = 0.0001; % tolerence
    batch_size = step_size;

    % note: there will be cnt+1 expectations

    % 1st batch
    x = log_growth(x0, r, batch_size);
    del_pt = ceil(batch_size*3/4);
    e(cnt) = mean(x(del_pt:end));

    % 2nd batch
    new_x0 = x(end);
    x = log_growth(new_x0, r, batch_size);
    e(cnt+1) = mean(x);

    % 1st difference
    d(cnt) = abs(e(cnt+1)-e(cnt));

    % while loop calculating differences
    while (true)
        % stopping criteria
        if (d(cnt) < tol)
            break
        elseif (mod(cnt, 10) == 0)
            batch_size = step_size*floor(cnt/10);
        end

        cnt = cnt + 1;
```

```
        new_x0 = x(end);
        x = log_growth(new_x0, r, batch_size);

        e(cnt+1) = mean(x);
        d(cnt) = abs(e(cnt+1)-e(cnt));
    end
    lims = uniquetol(x, tol);
end

function x = log_growth(x0, r, batch_size)
    x = zeros(1, batch_size);
    x(1) = recur(x0, r);
    for i = 2:batch_size
        x(i) = recur(x(i-1), r);
    end
end

function x1 = recur(x0, r)
    x1 = r*x0*(1-x0);
end
```

[5] mutual_info.m

```
function mi =  mutual_info(x, tau, nbins)
    % tau is multiplier of time-step of ODE solver; approximately 0.01 in default
    % nbin is number of bins

    [len, ~] = size(x);

    % extract x_tau and align sizes of x and x_tau
    x_tau = x(1+tau:len);
    x(len-(tau-1):end) = [];

    % probability distributions
    joint_freq_mx = hist3([x, x_tau], 'Nbins',[nbins, nbins]);
    pjoint = joint_freq_mx ./ sum(joint_freq_mx, 'all');

    xfreq = histcounts(x, nbins);
    px = xfreq ./ sum(xfreq);

    % entropy
    hx = -sum(px .* log(px));

    % conditional entropy
    [~, ncols] = size(joint_freq_mx);
    h_xly_elt = zeros(1, ncols);
    for i=1:ncols
        % p(x|y=x_tau(i))
```

```
        xly_freq = joint_freq_mx(:,i);
        p_xly = xly_freq ./ sum(xly_freq);

        % to avoid log(0); force them to be 0 (=log(1))
        p_xly(p_xly == 0) = 1;

        % p(x, y=x_tau(i))
        pjoint_col = pjoint(:,i);


        h_xly_elt(i) = sum(pjoint_col .* log(p_xly));
    end
    h_xly = -sum(h_xly_elt);

    mi = hx - h_xly;
end
```

[6] FNN.m

```
function [fnn_ratios] = FNN(x, tau, dim, r, tol)
    xlen = length(x);

    % 1st neighborhood info
    idxs = zeros(1, xlen);
    old_dists = zeros(1, xlen);
    for i=1:xlen
        entire_nbhd = x;
        entire_nbhd(i) = [];
        [kidx, kdist] = knnsearch(entire_nbhd, x(i), 'K',r, 'Distance','euclidean');

        if (kidx < i)
            idxs(i) = kidx(r);
        else
            idxs(i) = kidx(r) + 1;
        end

        old_dists(i) = kdist(r)^2;
    end

    fnn_ratios = zeros(1, length(dim));
    for m=1:length(dim)
        fprintf('Dimension %i \n', m)
        fnn_cnts = 0;

        % calculate new distances
        xlen_next = xlen - m*tau; % remove last a few elements out of range
        new_dists = zeros(1, xlen_next);
```

```
        for i=1:xlen_next
            new_idx = i + m*tau;

            entire_nbhd = x;
            entire_nbhd(new_idx) = [];
            [~, kdist] = knnsearch(entire_nbhd, x(new_idx), 'K',r, 'Distance','euclidean');

            new_dists(i) = old_dists(i) + kdist(r)^2;

            % FNN criterion (FNN percentage)
            crit = power((new_dists(i) - old_dists(i))/old_dists(i), 1/2);
            if (crit > tol)
                fnn_cnts = fnn_cnts + 1;
            end
        end
        fnn_ratios(m) = fnn_cnts / xlen_next;
        old_dists = new_dists;
    end
end
```