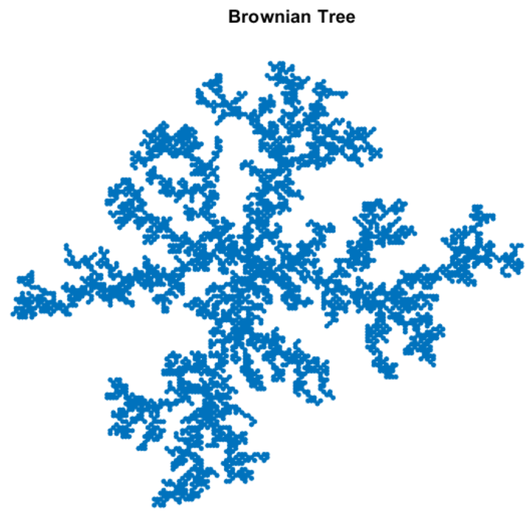
IE 597 Homework 3

Fall 2020

Haedong Kim

Problem 1. Brownian tree

A seed point was a center, (200, 200) in terms of index, of 400x400 size grid. The algorithm in Appendix - [2] was run until 15,000 points were plotted in the grid.



Problem 2. Fractal dimension

A fractal dimension is calculated as follow. Length of radius was considered as l, and the number of points in the Euclidean circle of the radius was defined by N.

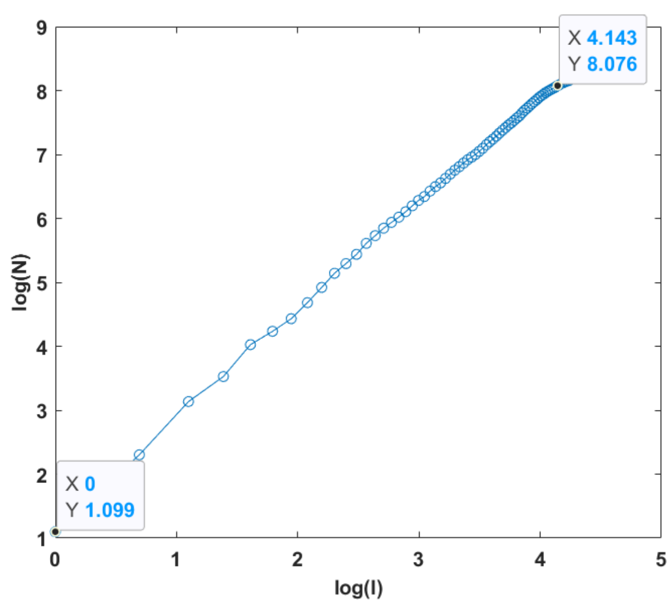


Figure 1. Relationship of N and l of the Brownian tree

Figure 1 shows the relation of $\log(N)$ and $\log(l)$ to investigate the fractal dimension $D \coloneqq \log(N)/\log(l)$. As expected, the slope, D, is almost constant and its value was 1.68.

Problem 3. Recurrence plots

Lorenz attractor of Figure 2 was taken to generate recurrence plots in Table 1. There are two recurrence plots: (a) is continuous version that show Euclidean distance between the points and (b) is the binary version of (a) with the $\epsilon = 5$.

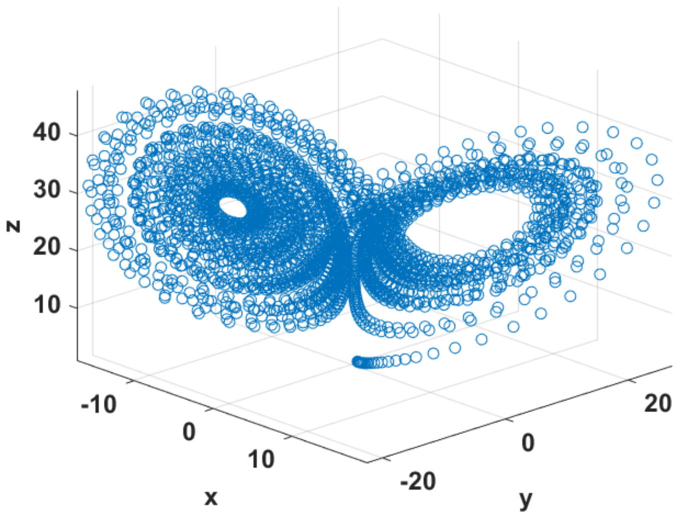
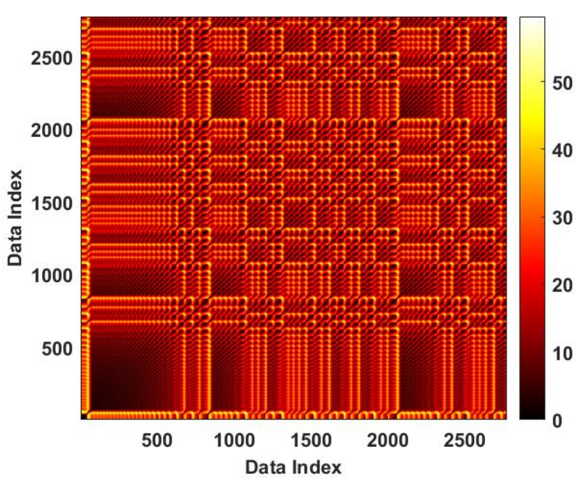


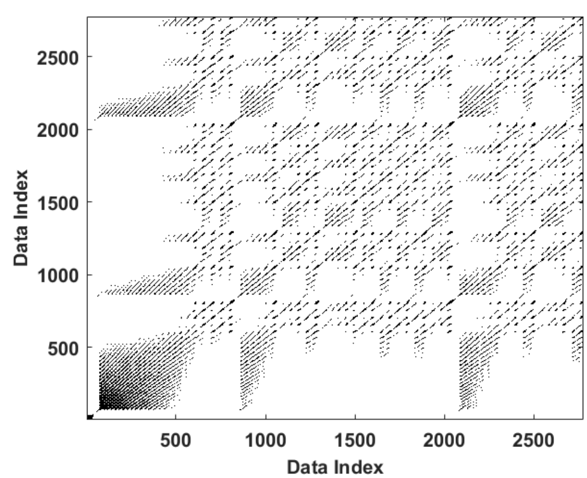
Figure 2. Lorenz attract with the initial point (0, 1, 1.05).

Table 1. Recurrence plots of the Lorenz attractor

(a) Continuous version



(c) Binary version



Problem 4. Recurrence quantification analysis (RQA)

You can fine MATLAB codes for recurrence rate, determinism, linemax, entropy, laminarity, and trapping time in Appendix – [3] to [8]. RQAs for the recurrence plot in Problem 3 are:

1. RR = 0.0635
2. DET = 0.9892 ($l_{min} = 200$)
3. LMAX = 382
4. ENT = 6.9434 ($l_{min} = 200$)
5. LAM = 0.9960 ($v_{min} = 100$)
6. TT = 986.0089 ($v_{min} = 100$)

Problem 5. Space time separation plot

x dimension of the Lorenz attractor used through this Homework was analyzed. Embedding dimension was chosen 2 and quantiles went from 0.1 to 0.9 by 0.1 increments. Figure 3 shows the resulting space time separation plot.

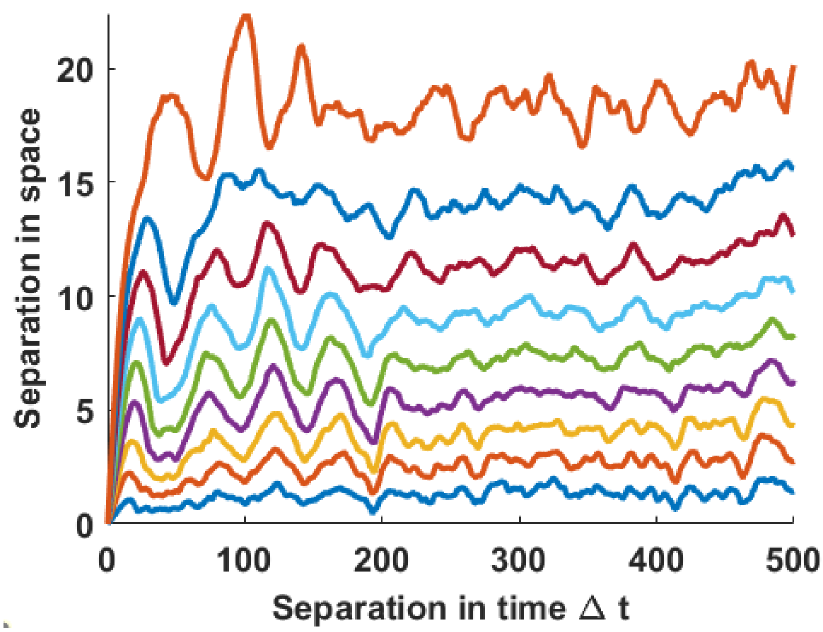


Figure 3. Space time separation plot of the Lorenz attractor

Appendix - MATLAB Code

[1] `HW3_haedong.m` (main script)

```
%% Problem 1 - Brownian tree
clc
close all
clear variables

n = 400;
```

```
bt_mx = BrownianTree2(n, 15000);
xx = [];
yy = [];

for i=1:n
    for j=1:n
        if (bt_mx(i,j) == 0)
            continue
        else
            xx = [xx, i];
            yy = [yy, j];
        end
    end
end

scatter(xx, yy, 10, 'filled')
set(gca,'XColor', 'none','YColor','none')
title('Brownian Tree')

save('BT.mat')

%% Problem 2 -  Box counting method
clc
close all
clear variables

load('BT.mat')
seed_idx = [200, 200];

% radius
rads = 1:99;
npts = zeros(1, length(rads));
for k=1:length(rads)
    r = rads(k);
    search_idx = -r:r;
    search_len = length(search_idx);

    cnt = 0;
    for i = 1:search_len
        for j = 1:search_len
            xidx = search_idx(i);
            yidx = search_idx(j);

            d = sqrt(xidx^2 + yidx^2);
            if (d<=r && d~=0)
```

```
                xidx = xidx + seed_idx(1);
                yidx = yidx + seed_idx(2);

                if(bt_mx(xidx, yidx))
                    cnt = cnt + 1;
                end
            else
                continue
            end
        end
    end
    npts(k) = cnt;
end

log_rads = log(rads);
log_npts = log(npts);

plot(log_rads, log_npts, '-o')
ylabel('log(N)')
xlabel('log(l)')

%% Problem 3 - Recurrence plot of Lorenz attractor
clc
close all
clear variables

lparam0 = [10, 28, 8/3];
linit = [0, 1, 1.05];
[t, s] = Lorenz(lparam0, linit, 50);
x = s;

xlen = length(x);
rec_mx = zeros(xlen);
for i=1:xlen
    for j=i:xlen
        d = pdist([x(i,:); x(j,:)], 'euclidean');
        rec_mx(i,j) = d;
        rec_mx(j,i) = d;
    end
end

% continuous version
figure(1)
imagesc(rec_mx);
colormap hot;
```

```
colorbar;
xlabel('Data Index')
ylabel('Data Index')
axis image;
get(gcf,'CurrentAxes');
set(gca,'YDir','normal')

% binary version
epsilon = 5;
bi_rec_mx = rec_mx <= epsilon;

figure(2)
imagesc(bi_rec_mx);
colormap([[1 1 1]; [0 0 0]])
xlabel('Data Index')
ylabel('Data Index')
axis image;
get(gcf,'CurrentAxes');
set(gca,'YDir','normal')

save('RP.mat')

%% Problem 4 - Recurrence quantification analysis
clc
close all
clear variables

load('RP.mat')

RR = recur_rate(bi_rec_mx);
DET = determinism(bi_rec_mx, 200);
LMAX = linemax(bi_rec_mx);
ENT = entropy(bi_rec_mx, 200);
LAM = laminarity(bi_rec_mx, 100);
TT = trap_time(bi_rec_mx, 100);

%% Problem 5 - Space time separation
clc
close all
clear variables

lparam0 = [10, 28, 8/3];
linit = [0, 1, 1.05];
[t, s] = Lorenz(lparam0, linit, 50);
```

```
x = s(:,1);

p = 0.1:0.1:0.9;
STPs = cell(1, length(p));
for i=1:length(p)
    STPs{i} = STP(x, 2, 500, p(i));
end

deltats = 0:1:500;
for j=1:length(p)
    hold on
    plot(deltats, STPs{j}, 'LineWidth',2)
    hold off
end
axis tight
xlabel('Separation in time \Delta t')
ylabel('Separation in space')
% legend('0.1','0.2','0.3','0.4','0.5','0.6','0.7','0.8','0.9')
```

[2] BrownianTree.m

```
function [bt_mx] = BrownianTree(n, npts)
    bt_mx = zeros(n, n);

    % seed in center
    x = floor(n/2);
    y = floor(n/2);
    bt_mx(x,y) = 1;

    for i=1:npts
        % sample new point
        if (i>1)
            x = datasample(1:n, 1, 'Replace',false);
            y = datasample(1:n, 1, 'Replace',false);
        end

        while (1)
            ox = x;
            oy = y;
            x = x + datasample(-1:1, 1, 'Replace',false);
            y = y + datasample(-1:1, 1, 'Replace',false);

            if (x<=n && y<=n && x>=1 && y>=1 && bt_mx(x,y))
                if (ox<=n && oy<=n && ox>=1 && oy>=1)
                    bt_mx(ox,oy) = 1;
                    break
```

```
                end
            end
            if (~(x<=n && y<=n && x>=1 && y>=1))
                break
            end
        end
    end
end
```

[3] recur_rate.m

```
function RR = recur_rate(RP)
    N = length(RP);
    RR = sum(RP, 'all') / (N^2);
end
```

[4] determinism.m

```
function DET = determinism(RP, lmin)
    N = length(RP);
    flip_RP = fliplr(RP);

    lprob = zeros(1, N);
    % diagonals (two on RP) length from 1 to N-1
    for i=1:(N-1)
        lprob(i) = i * (sum(diag(flip_RP, i), 'all') + sum(diag(flip_RP, -
i), 'all')) / sum(RP, 'all');
    end

    % main diagonal (one on RP)
    lprob(N) = N * sum(diag(flip_RP, 0)) / sum(RP, 'all');

    DET = sum(lprob(lmin:end)) / sum(lprob);
end
```

[5] linemax.m

```
function LMAX = linemax(RP)
    N = length(RP);
    flip_RP = fliplr(RP);

    l = zeros(1, N);
    for i=1:(N-1)
        l(i) = (sum(diag(flip_RP, i), 'all') + sum(diag(flip_RP, -i), 'all'));
    end
    l(N) = sum(diag(flip_RP, 0));
```

```
    LMAX = max(l);
end
```

[6] entropy.m

```
function ENT = entropy(RP, lmin)
    N = length(RP);
    flip_RP = fliplr(RP);

    lprob = zeros(1, N);
    % diagonals (two on RP) length from 1 to N-1
    for i=1:(N-1)
        lprob(i) = (sum(diag(flip_RP, i), 'all') + sum(diag(flip_RP, -
i), 'all')) / sum(RP, 'all');
    end

    % main diagonal (one on RP)
    lprob(N) = sum(diag(flip_RP, 0)) / sum(RP, 'all');

    ENT = -sum(lprob(lmin:end) .* log(lprob(lmin:end)));
end
```

[7] laminarity.m

```
function LAM = laminarity(RP, vmin)
    N = length(RP);

    vprob = zeros(1, N);
    for i=1:N
        vprob(i) = i * sum(RP(i:N,:), 'all') / sum(RP, 'all');
    end

    LAM = sum(vprob(vmin:end)) / sum(vprob);
end
```

[8] trap_time.m

```
function TT = trap_time(RP, vmin)
    N = length(RP);
    v = vmin:1:N;

    prob = zeros(1, N);
    for i=1:N
        prob(i) = sum(RP(i:N,:), 'all') / sum(RP, 'all');
    end

    TT = sum(v .* prob(vmin:end)) / sum(prob(vmin:end));
End
```

[9] STP.m

```
function stp_data = STP(x, m, max_deltat, p)
    xlen = length(x);

    % 1st dimension
    x1 = x(m:xlen);

    % 2nd dimension
    x2 = x(1:(xlen-m+1));

    X = [x1, x2];
    Xlen = length(X);

    deltats = 0:1:max_deltat;

    stp_data = zeros(1, length(deltats));
    for i=1:length(deltats)
        deltat = deltats(i);

        n = 1:1:(Xlen-deltat);
        dnt = zeros(length(n), 1);
        for j=1:length(n)
            dnt(j) = pdist([x(j,:); x(j+deltat,:)], 'cityblock');
        end

        Qp = floor(p*(Xlen-deltat));

        dnt_order = sort(dnt, 'ascend');
        stp_data(i) = dnt_order(Qp);
    end
end
```