Homework Assignment № 8

Kevin Mack, Clarkson University 12/14/2017

Question 1:

In the first task of the assignment, there is Matlab code provided from section A3 of the appendix from *Applied and Computational Measurable Dynamics* by Erik M. Bollt and Naratip Santitissadeekorn. The code is used to perform several tasks on data sets from chaotic systems. The tasks include Delaunay Triangulation, and use of the Ulam-Galerkin matrix to estimate the Frobenius-Perron operator. The methods will be used on a Lorenz attractor given by (1), the standard map in (2), and finally the Rossler attractor in (5).

The Lorenz attractor is a system of ordinary differential equations which produce chaotic solutions for certain parameter values of parameters (σ, ρ, and β) and initial conditions,

$$\begin{aligned} \frac{dx}{dt} &= \sigma(y - x), \\ \frac{dy}{dt} &= x(\rho - z) - y, \\ \frac{dz}{dt} &= xy - \beta z. \end{aligned} \tag{1}$$

The code to produce the Delaunay triangulation of the Lorenz attractor is given in appendix A.3 (**runnerLorenzCover.m**) and is shown in the first code listing.

Listing 1: Matlab code – runnerLorenzCover.m

```
clear; close all;
load 'LorenzDat.mat' %This should be n x 3 array of real numbers in this case being points on (near) the lorenz attractor
figure
hold on
grid on
grid minor
plot3(X(:,1),X(:,2),X(:,3),'.')
z=X;

xlow=floor(min(X)); xhigh=ceil(max(X)); h=2.5;
[X1,X2,X3] = ndgrid(xlow(1):h:xhigh(1), xlow(2):h:xhigh(2), xlow(3):h:xhigh(3));
m=size(X1);

x1=reshape(X1,m(1)*m(2)*m(3),1); x2=reshape(X2,m(1)*m(2)*m(3),1); x3=reshape(X3,m(1)*m(2)*m(3),1);
```

```
15
16
17 %Formulate Delaunay Triangulation of region
18 dt= DelaunayTri([x1 x2, x3]);   %See Matlab Subroutine DelauneyTri for ↩
       input/output information
19                              %dt is the triangulation class
20                              %
21                              %   where
22                              %
23                              %dt.dt.Triangulation is an m1 by 4 array of
24                              %integers labelling the vertex corner numbers
25                              %of the triangles
26                              %
27                              %   and
28                              %
29                              %dt.X is an m2 by 3 array of real numbers
30                              %defining positions
31
32
33
34 %Count number of orbit points in z which cause a triangle to be counted as
35 %occupied (and otherwise a triangle is not counted as it is empty until
36 %observed occupied
37
38 nottrue=0;
39 while(nottrue<1)
40     nottrue=1;
41     SI = pointLocation(dt,z); %Matlab command, Locate the simplex element ↩
           in dt
42                               %containing the specified locations of each
43                               %of the elements of the array z of orbit
44                               %samples
45                               %
46     l=unique(SI);  %Matlab subrouting: Count the number of unique ↩
           instances in SI
47     k=1;
48     while(isnan(l(k))<1&&k<length(l)) %Using Matlab subroutine True for ↩
           Not-a-Number
49         [ii,j]=find(SI==l(k)); %Collect those locations corresponding to ↩
               each unique l.
50         cnt(k)=sum(j);
51         k=k+1;
52     end
53     k=k-1;    ll=l(1:k); cnt=reshape(cnt,size(ll));
54 end
55 %%
56
```

```
% Plot those  simplex elements of the dt which are occupied by an orbit
% iterate of z - dt(ll,:) are those occupied simplex elements
figure;
hold on
grid on
grid minor
plot3(0,0,0); patch('faces',dt(ll,:), 'vertices', dt.X, 'FaceColor','r');
N=length(ll);

%Creating and Editing Delaunay Triangulations
```

The code called **runnerLorenzCover.m** produces Figure 1, which shows several angles of the Lorenz attractor as it is covered by a Delaunay triangulation algorithm. The algorithm can be adjusted based on how much resolution is needed for the cover (by changing the tessellation size, h).

Further, the Ulam-Galerkin matrix is used to estimate the Frobenius-Perron operator and will be used to evaluate the standard map. The standard map is an area-preserving chaotic map, where p_n and θ_n are taken modulo 2π,

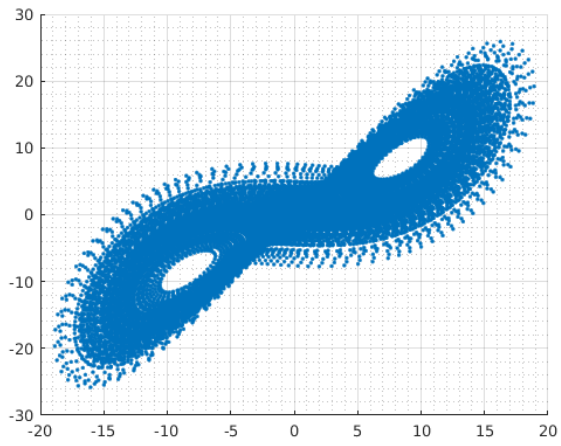
$$\begin{aligned} p_{n+1} &= p_n + Ksin(\theta_n), \\ \theta_{n+1} &= \theta_n + p_{n+1}. \end{aligned} \tag{2}$$

The code to produce the standard map is given in the listing below, and can be found in the appendix A.1.2.

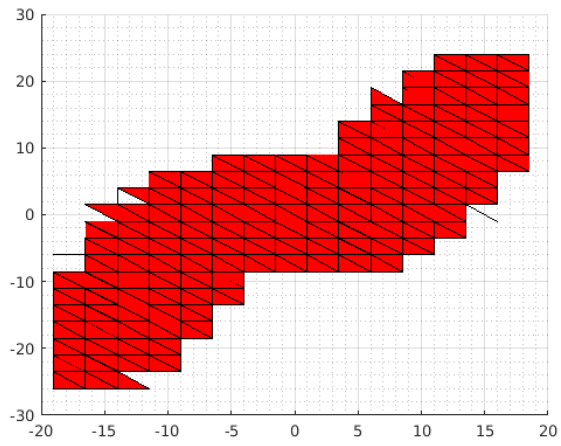
Listing 2: Matlab code – standard.m

```
function xvecNew = standard(xvec,k)
% Matlab coded in vectorized form of standard
%Input:
%       xvec - vector of initial conditions
%       k    - standard map parameter
%
%Output:
%       xvecNew - vector of their images
%
xvecNew = zeros(size(xvec));
xvecNew(1,:) = xvec(1,:)+xvec(2,:)-k*sin(2*pi*xvec(1,:))/(2*pi);
xvecNew(2,:) = xvec(2,:)-k*sin(2*pi*xvec(1,:))/(2*pi);
```

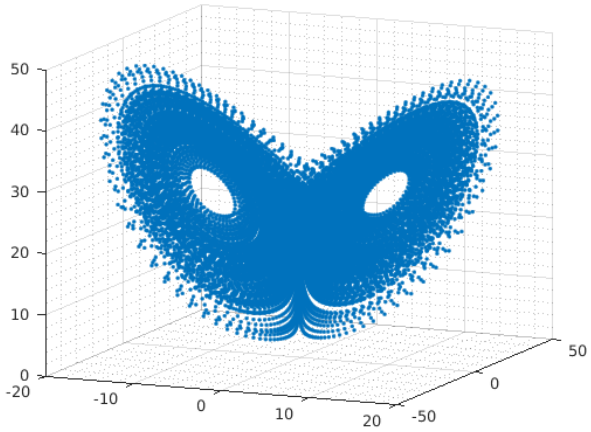
The code to create the stochastic matrix, $\mathbf{A}$, of the standard map is called **TransitionMatrix.m** and is tested by **runnerSimpleTriCover.m** (appendix A.1.2), which are both given in the listings below. The **runnerSimpleTriCover.m** code makes a sample orbit of the standard map, then calls **TransitionMatrix.m** to use the Ulam-Galerkin method to approximate the Frobenius-Perron operator in phase space after being thresholded with second eigenvector, v_2. The output



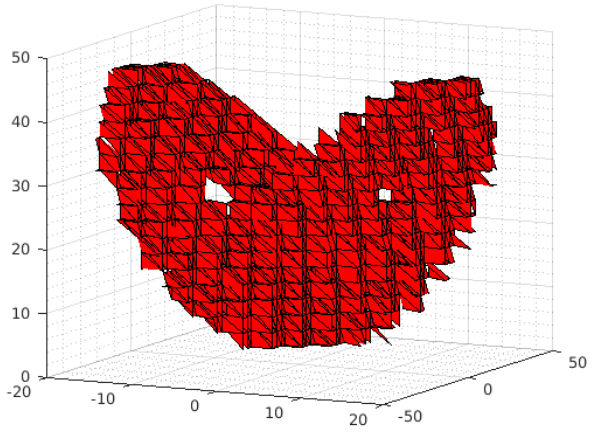
(a) Angle 1, Lorenz Attractor



(b) Angle 1, Lorenz Attractor outer cover $h = 2.5$



(c) Angle 2, Lorenz Attractor



(d) Angle 2, Lorenz Attractor outer cover $h = 2.5$

Figure 1: The images in the left column show plotted points of the Lorenz attractor for a specific set of parameters and initial conditions. The images in the right column represent the outer cover of the attractor which has been achieved through Delaunay triangulation.

from the code can be seen for different values of k in Figures 2 and 3 in the next section.

Listing 3: Matlab code – runnerSimpleTriCover.m

```
close all; clear;
nPlot = 10000;   % nPlot-iterate test orbit this many points after the transient.
M=250; diam=10; a=0; b=1;
h=0.025

%%
%%%%%%%Make a Sample Orbit of the Standard Map
k=1.2; %k is chosen >0.97... for the magic breakup of the "golden mean" torus
z=[];
```

```
10 % Set some variables
11 transientSteps = 100;   %length of initial transient of test orbit to ↩
       ignore
12 initialX = rand(2,1);
13 x = initialX; hold on;
14
15 %%
16 % Throw away transients:
17 for i=1:transientSteps
18     x = standard(x,k); %Use as a test example, the standard map.
19 end
20
21 %%
22 % Plot the next nPlot points visited:
23 for i=1:nPlot
24     x = standard(x,k);
25     z=[z;x'];
26 end
27 z=mod(z,1);
28 plot(z(:,1),z(:,2),'b.','markersize',10); hold on; %See Fig. 11.1.
29 %
30 %%
31
32
33 %%%%%%%%%%%%%%%
34 %%%Build an Ulam-Galerkin's Matrix Based on a Test Orbit Visiting ↩
       Triangles
35 %%%of a Tesselation
36 [dt,ll,A,zz]=TransitionMatrix(z,h,a,b,a,b);
37 triplot(dt) %Matlab routine to draw the triangulation simplex
38             %See Fig. 11.1.
39 drawnow;
40
41 %%
42 [v,d]=eigs(A',2); w=abs(v(:,1)); %Compute 1st and send eigenvalues/vectors↩
       of Galerkin-Ulam Matrix A
43 %
44 figure; stem3(zz(:,1),zz(:,2),w(:),'fill'); %Show the dominant eigenvector↩
       (d(1)=1) meant to roughly estimate invariant density
45 %
46 w2=v(:,1); [i,ww]=find(w2>0);
47 figure; stem3(zz(i,1),zz(i,2),w2(i),'r', 'fill')  %See Fig. 11.2.
48
49 %% Produce a reversible Markov Chain R
50 P=A;
51 [v,d]=eigs(A',1);
52 N=size(A,1);
```

```
for i=1:N
    for j=1:N
        Phat(i,j)=v(j,1)*P(j,i)/v(i,1);
    end
end
R=(P+Phat)/2;

%%
%Partition
[w,l]=eigs(R,4);
figure; plot(w(:,2)) %See Fig. 11.3
c=0; eps=0.005;
[i,ww]=find(w(:,2)>c); [ii,ww]=find(w(:,2)<=c); [iii,ww]=find(abs(w(:,2))<eps);

%% These commands plot the resulting partition in phase space.
%  See Fig. 11.4.
figure; patch('faces',dt(ll(i),:), 'vertices', dt.X, 'FaceColor','r'); hold on;
patch('faces',dt(ll(ii),:), 'vertices', dt.X, 'FaceColor','b');
patch('faces',dt(ll(iii),:), 'vertices', dt.X, 'FaceColor','k');
```

Listing 4: Matlab code – TransitionMatrix.m

```
%%%%%%%%%%%%%%%%
%% by Erik Bollt
%%%Build an Ulam-Galerkin's Matrix Based on a Test Orbit Visiting Triangles
%%%of a Tesselation
%%%%%%%%%%%%%%%%
% Input:
%       z - Test orbit is n x 2 for an n-iterate orbit sample
%       h - is the side length of the triangle edges which share a right
%       angle
%       ax, bx, ay, by - the low and high ends of a box bounding data
% Output:
%       dt - a DelaunayTri
%%

function [dt,ll,A,zz]=TransitionMatrix(z,h,ax,bx,ay,by)

%%

%low=-2; high=2; low=0; high=1;
[X1,X2] = ndgrid(ax:h:bx, ay:h:by);
[m,n]=size(X1);
```

```
22 x1=reshape(X1,m*n,1); x2=reshape(X2,m*n,1);
23
24 %Formulate Delaunay Triangulation of region
25 dt= DelaunayTri([x1 x2]);   %See Matlab Subroutine DelauneyTri for input/↩
       output information
26                             %dt is the triangulation class
27                             %
28                             %   where
29                             %
30                             %dt.dt.Triangulation is an m1 by 3 array of
31                             %integers labelling the vertex corner numbers
32                             %of the triangles
33                             %
34                             %   and
35                             %
36                             %dt.X is an m2 by 2 array of real numbers
37                             %defining positions
38
39 %triplot(dt) %Optional plot command of this triangulation
40
41 %%
42
43
44 %Count number of orbit points in z which cause a triangle to be counted as
45 %occupied (and otherwise a triangle is not counted as it is empty until
46 %observed occupied
47
48 nottrue=0;
49 while(nottrue<1)
50     nottrue=1;
51     SI = pointLocation(dt,z); %Matlab command, Locate the simplex element ↩
           in dt
52                               %containing the specified locations of each
53                               %of the elements of the array z of orbit
54                               %samples
55                               %
56     l=unique(SI);  %Matlab subrouting: Count the number of unique ↩
           instances in SI
57     k=1;
58     while(isnan(l(k))<1&&k<length(l)) %Using Matlab subroutine True for ↩
           Not-a-Number
59         [ii,j]=find(SI==l(k)); %Collect those locations corresponding to ↩
               each unique l.
60         cnt(k)=sum(j);
61         k=k+1;
62     end
63     k=k-1;    ll=l(1:k); cnt=reshape(cnt,size(ll));
```

```
64 end
65 %%
66
67 % Plot those  simplex elements of the dt which are occupied by an orbit
68 % iterate of z - dt(ll,:) are those occupied simplex elements
69 patch('faces',dt(ll,:), 'vertices', dt.X, 'FaceColor','r');
70 N=length(ll);
71
72 %%
73
74 %Translateback from ll back to phase space positions z by using the center
75 %positions with Matlab subroutine "mean"
76 zz=zeros(N,2);
77 for i=1:N
78     zz(i,:)=mean([dt.X(dt.Triangulation(ll(i),1),:)
79     dt.X(dt.Triangulation(ll(i),2),:)
80     dt.X(dt.Triangulation(ll(i),3),:)]);
81 end
82
83 %%Now build the transition matrix A
84 %
85 %So that A(i,j)>0 iff there is an element in simplex element i such that ↩
      there is an iterate z(k,:) and that
86 % the next  iterate, z(k+1,:), transitions to simplex element j
87 A=zeros(N,N);
88 for i=1:(length(SI)-1)
89     ii= find(ll==SI(i));
90     jj=find(ll==SI(i+1));
91     %[ii jj size(A)]
92     A(ii,jj)=A(ii,jj)+1;
93 end
94 %Now make A into a stochastic matrix by row normalizing
95 for i=1:N
96     q=length(find(SI==ll(i)));
97     A(i,:)=A(i,:)./q;
98     A(i,:)=A(i,:)./sum(A(i,:));
99 end
```

Question 2:

In Question 2, the task is to investigate the almost invariant sets of the standard map as the critical parameter, k, is stepped from 0.9 to 1.5. The standard map is given by (2), and is known to be non-linear (for $k \neq 0$) and chaotic. The graphs in Figure 2 show similar properties, as compared with the graphs in Figure 3. It should be noted that for a value of $k = 0$ the standard map

is linear, and increases in non-linearity as k is increased. The graphs were created with code from Question 1 (**standard.m**, **runnerSimpleTriCover.m**, and **TransitionMatrix.m**). According to the reference material *A Glance at the Standard Map* by Ryan Tobin, as the non-linearity factor, k, is increased, the nonintegrable components of the standard map become more apparent. Also, the final invariant torus is terminated when the irrationality of the winding number corresponding to the last quasiperiodic orbit is farthest from rational, which occurs at the golden mean (approximately 1.61). This occurs at $k > 0.97$ and can be seen by comparing the graphs in the left column of Figures 2 and 3, which include the standard map from $k = 0.9$ to $k = 1.5$. In summary, for each different value of k, the codes work together to produce an Ulam-Galerkin estimate of the Frobenius-Perron operator. The major steps of the code are as follows:

Step 1: Create a test orbit of the standard map (**standard.m** and **runnerSimpleTriCover.m**)

Step 2: Perform a Delaunay triangulation in the specified domain for the test orbit for a set k-value (**runnerSimpleTriCover.m** and built-in function **DelaunayTri.m**)

Step 3: Prune the initial triangulation to a smaller collection of triangles which include only those triangles which have been visited by the test orbit. (**TransitionMatrix.m**)

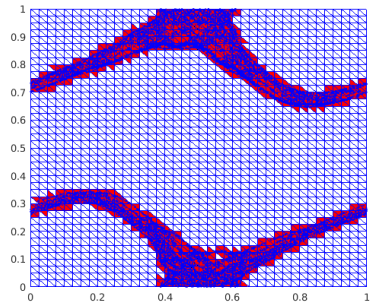
Step 4: Form stochastic matrix, $\mathbf{A}$ (which is an $M \times M$ matrix where M is the smaller number of triangles), by using the Ulam-Galerkin estimate of the Frobenius-Perron operator. **TransitionMatrix.m**)

The Frobenius-Perron operator can be defined as an infinitely large stochastic matrix acting on an infinite-dimensional linear space, however this representation is not particularly useful in practice. Instead, a finite-rank approximation of the operator is obtained through the process described above. The matrix approximation we are after is given by,

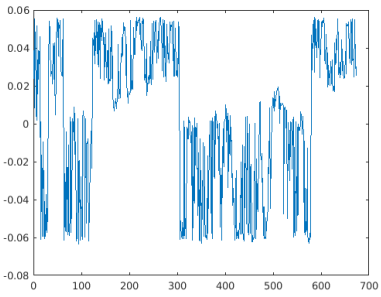
$$P_{i,j} = \frac{m(B_i \cap F^{-1}(B_j))}{m(B_i)}, \tag{3}$$

where m denotes the Lebesgue measure on M. However, since we are working with a test orbit (4) then becomes,

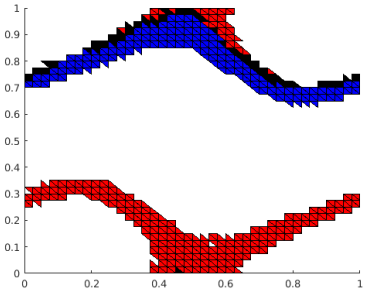
$$P_{i,j} \approx \frac{\#(\{x_k | x_k \in B_i \text{ and } F(x_k) \in B_j\})}{\#(\{x_k \in B_i\})}, \tag{4}$$



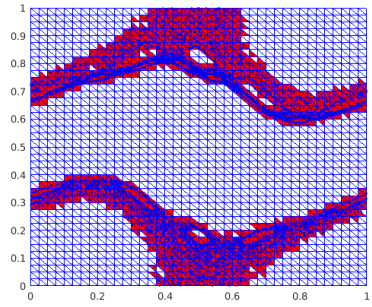
(a) A Delaunay triangulation for the test orbit, $k = 0.9$



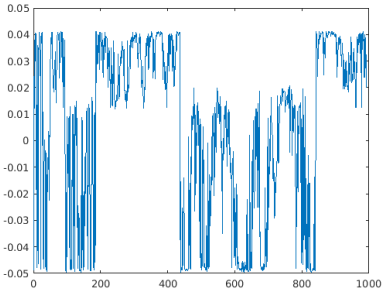
(b) The eigenvector v_2 for $k = 0.9$



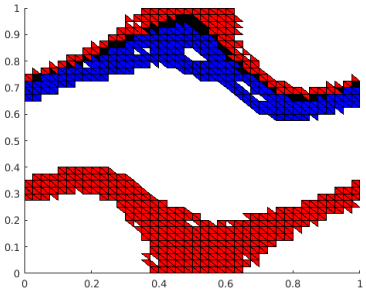
(c) The thresholded weakly invariant sets of the test orbit for $k = 0.9$



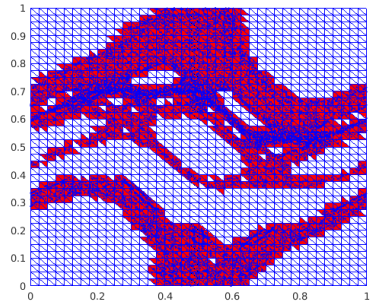
(d) A Delaunay triangulation for the test orbit, $k = 1.0$



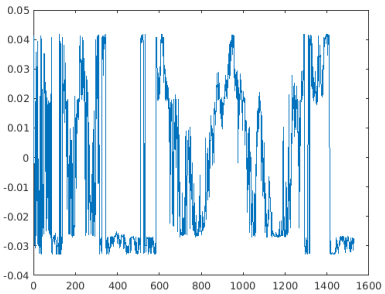
(e) The eigenvector v_2 for $k = 1.0$



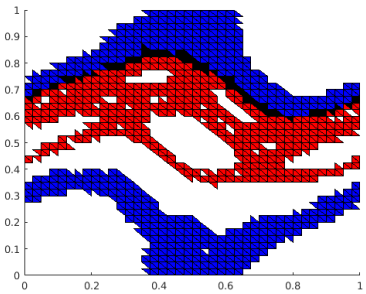
(f) The thresholded weakly invariant sets of the test orbit for $k = 1.0$



(g) A Delaunay triangulation for the test orbit, $k = 1.1$

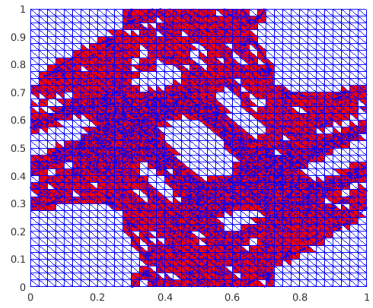


(h) The eigenvector v_2 for $k = 1.1$

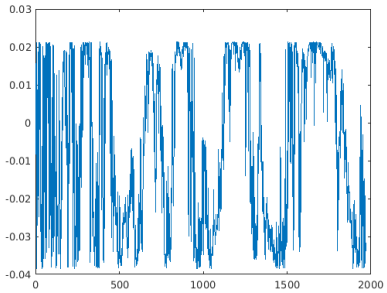


(i) The thresholded weakly invariant sets of the test orbit for $k = 1.1$

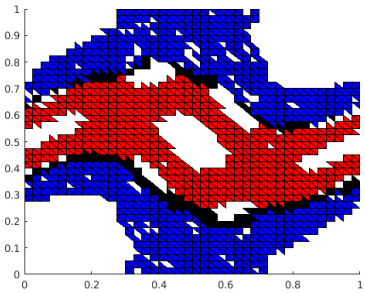
Figure 2: The images in the left column represent the Delaunay triangulation of the standard map. The blue dots represent the test orbit, while the red triangles are the grid elements visited by the test orbit. The images in the middle column are the second eigenvector, v_2, of the reversible Markov chain. Lastly, the images in the right column are the almost invariant sets of the standard map test orbit on the pruned tessellation (shown in the left column). The eigenvector v_2 can be thresholded to produce the tessellation elements corresponding to weakly transitive comnponents (red and blue), with the triangles on the boundary (black).



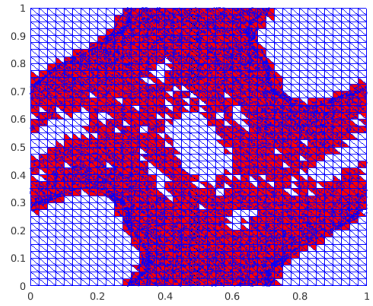
(a) A Delaunay triangulation for the test orbit, $k = 1.2$



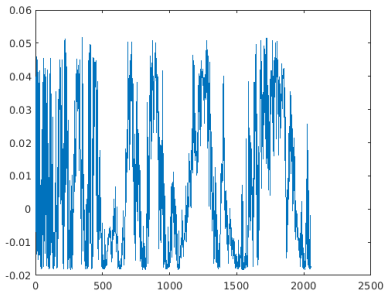
(b) The eigenvector v_2 for $k = 1.2$



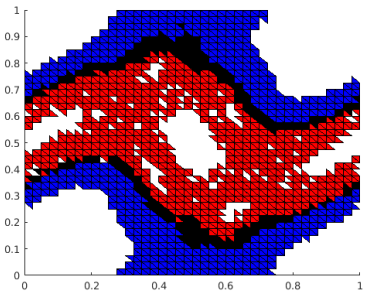
(c) The thresholded weakly invariant sets of the test orbit for $k = 1.2$



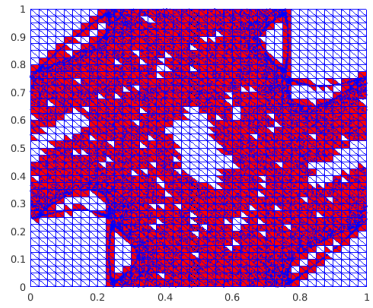
(d) A Delaunay triangulation for the test orbit, $k = 1.4$



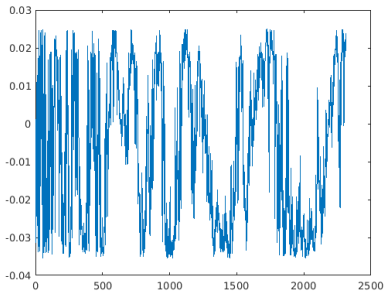
(e) The eigenvector v_2 for $k = 1.4$



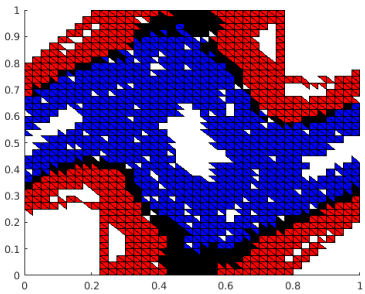
(f) The thresholded weakly invariant sets of the test orbit for $k = 1.4$



(g) A Delaunay triangulation for the test orbit, $k = 1.5$



(h) The eigenvector v_2 for $k = 1.5$



(i) The thresholded weakly invariant sets of the test orbit for $k = 1.5$

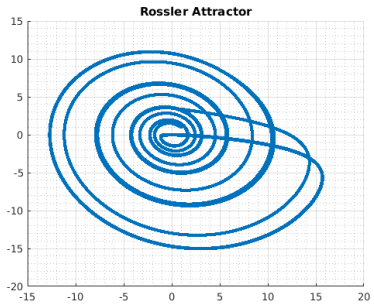
Figure 3: The images here represent the same concepts as in Figure 2, but for values of $k = \{1.2, 1.4, 1.5\}$. It can be seen here in the middle column of images of eigenvector v_2 that the higher the value of k, the more the oscillation is present between sets (or sets). It can also be seen that the higher k-values produce more chaotic behaviour, as is expected. (Note that graphs for $k = 1.3$ have been omitted simply to save space, since their behaviour is similar to the other graphs present)

Question 3:

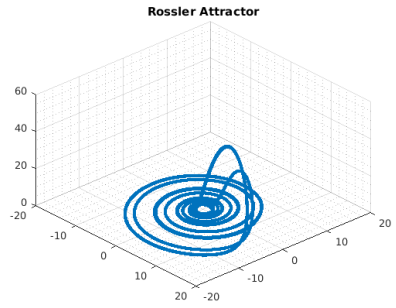
In this section the task is to apply the same methods in Part 1 to the Rossler Attractor (instead of the Lorenz attractor). The Matlab code will attempt to create an outer cover of the chaotic attractor. The Rossler Equations are given in (5), with parameters $a = 0.2$, $b = 0.2$, and $c = 8.0$.

$$\begin{aligned} \frac{dx}{dt} &= -y - z \\ \frac{dy}{dt} &= x + ay \\ \frac{dz}{dt} &= b + z(x - c) \end{aligned} \tag{5}$$

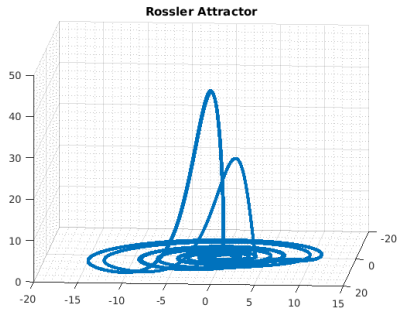
The Rossler attractor is modeled by the code **rossler.m**, which makes use of Matlab's **ode45** to solve a set of ordinary differential equations. The data from rossler.m is then used by **runnerRosslerCover.m** to create a cover of the attractor (similar to **runnerLorenzCover.m** in Question 1). The results, using the same Delaunay Triangulation parameters as in Question 1, are given by Figure 4.



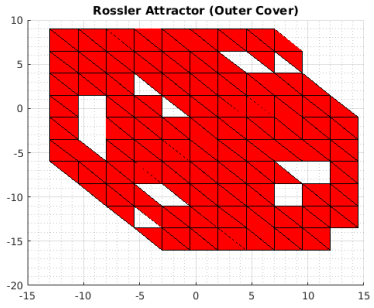
(a) Angle 1, Rossler Attractor



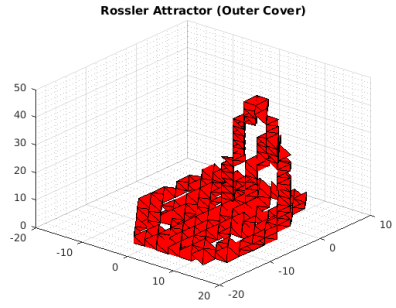
(b) Angle 2, Rossler Attractor



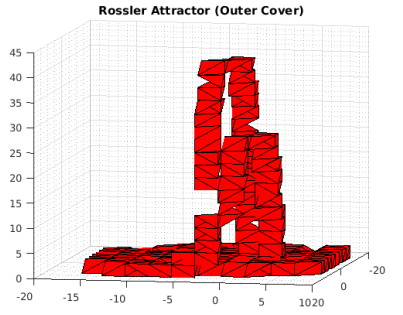
(c) Angle 3, Rossler Attractor



(d) Angle 1, Rossler Attractor outer cover $h = 2.5$



(e) Angle 2, Rossler Attractor outer cover $h = 2.5$



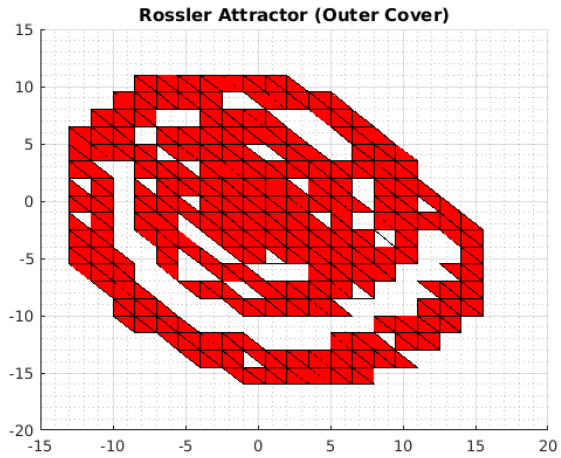
(f) Angle 3, Rossler Attractor outer cover $h = 2.5$

Figure 4: The figure depicts the accuracy of computing the outer cover of the Rossler attractor using Delaunay triangulation and basic grid element size $h = 2.5$. It is clear from the graphs that the grid elements are too large to give a proper representation of the outer cover, causing the figures on the bottom row to appear to be a very low representation of the attractor. This can be fixed by adjusting the h parameter, and the results of doing so are represented in Figure 5.

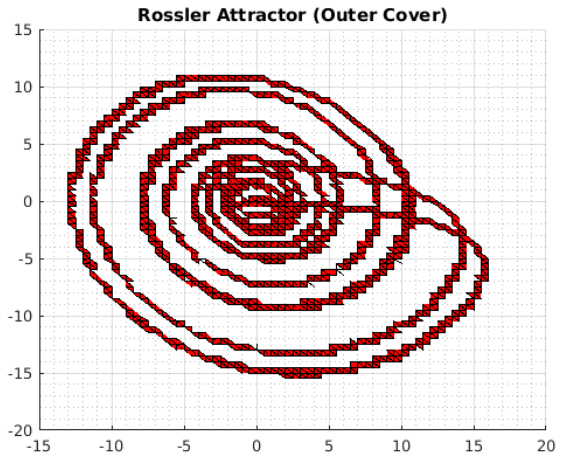
Listing 5: Matlab code – rossler.m

```
%% Create Data for Rossler Equations
clear;
a = 0.2; b = 0.2; c = 8.0; % set parameters for Rossler attractor
t = 0:.0001:100; % set time step and interval

x_init = 1; y_init = 1; z_init = 1; % set initial conditions

% Rossler ODE
f = @(t,x) [ -x(2)-x(3); x(1)+a*x(2); b+x(1)*x(3)-c*x(3)];

[t1,x1] = ode45(f, t, [x_init y_init z_init]);

% plot results
figure
hold on
grid on
grid minor
title('Rossler Equations')
plot3(x1(:,1), x1(:,2), x1(:,3), 'k-')

% Save Rossler equations data
X = x1;
X_data = X';
filename = 'rossler.mat';
save(filename, 'X', 'X_data');
```

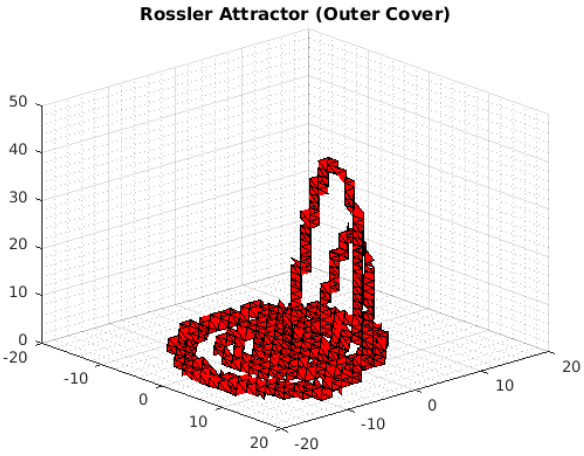
It can be seen that in the Figure 4, the tessellation size is not low enough to give a very descriptive cover for this particular attractor. This can be fixed by adjusting the size of the elements (decreasing the tessellation size). The results from this adjustment can be seen in Figure 5. The desired accuracy can be achieved through this adjustment (tessellation size h), with the caveat that the size of the sparse matrix, $\mathbf{A}$, scales in size according to $\left(\frac{1}{h}\right)^d$ for domain dimension d (meaning more computation time and resources).



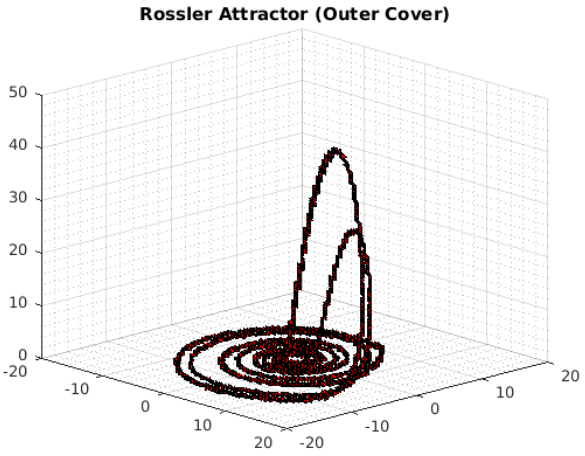
(a) Angle 1, Rossler Attractor outer cover $h = 1.5$



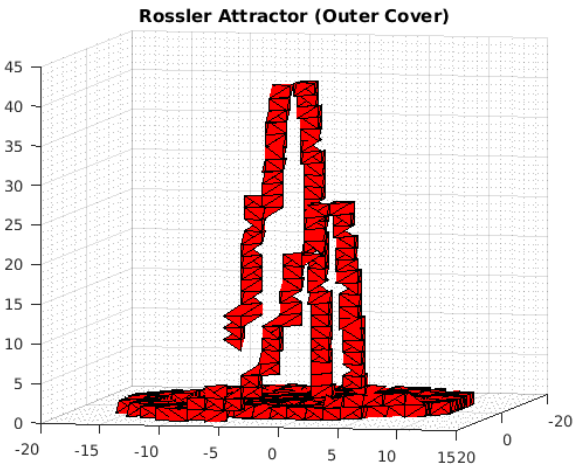
(b) Angle 1, Rossler Attractor outer cover $h = 0.5$



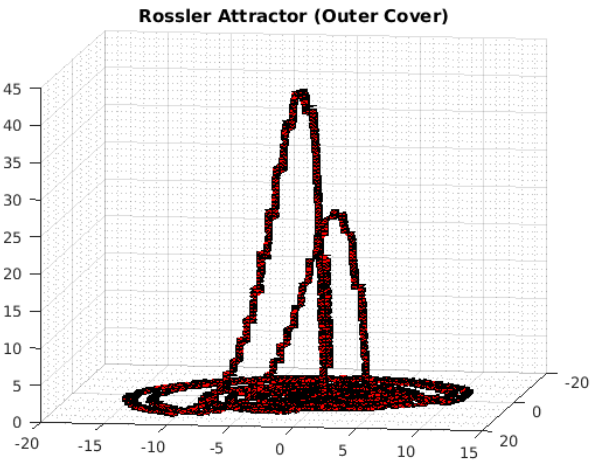
(c) Angle 2, Rossler Attractor outer cover $h = 1.5$



(d) Angle 2, Rossler Attractor outer cover $h = 0.5$



(e) Angle 3, Rossler Attractor outer cover $h = 1.5$



(f) Angle 3, Rossler Attractor outer cover $h = 0.5$

Figure 5: Results of varying the parameter h in the cover computation. The outer cover of the Rossler attractor becomes more clear as h is decreased, however this requires more computation resources. The h parameter corresponds to the tessellation size. The solution with the best accuracy (of the tested h values) is when $h = 0.5$. This attractor required more resolution to achieve results that are visibly satisfying than the Lorenz attractor from Question 1.