ESE 5582 Homework 2 / Frobenius-Perron Operator / Koopman Operator Approaches

Seif Elkhashab / 02-29-2024 /

1. Consider the classical Van der Pol system:

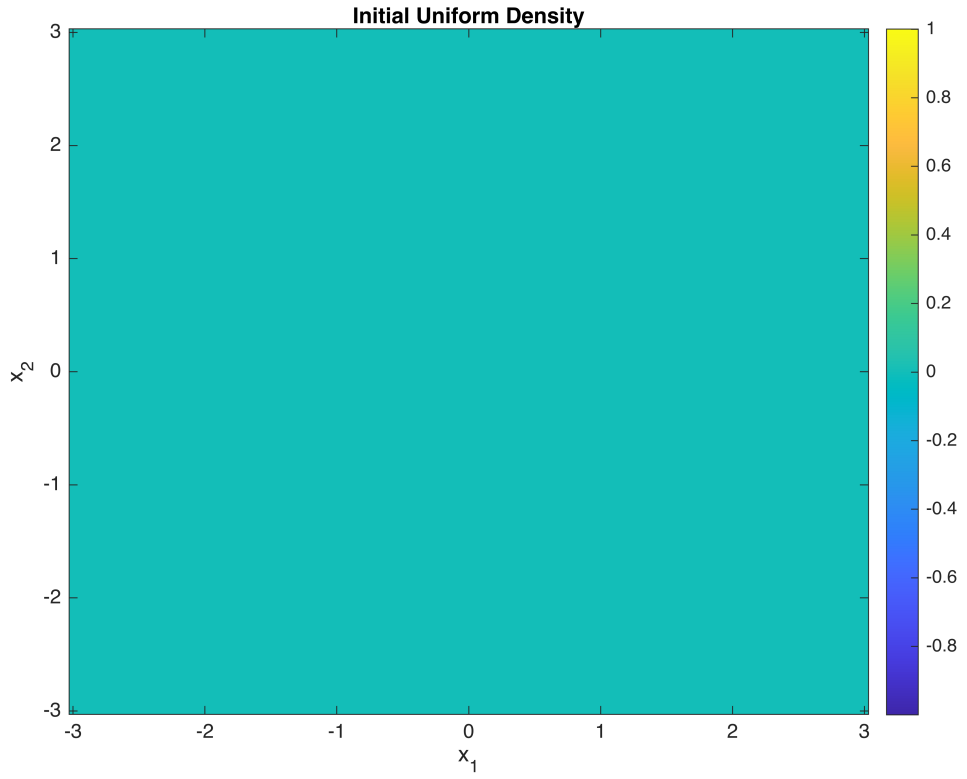
$$\dot{x}_1 = x_2,$$

$$\dot{x}_2 = (1 - x_1^2)x_2 - x_1$$

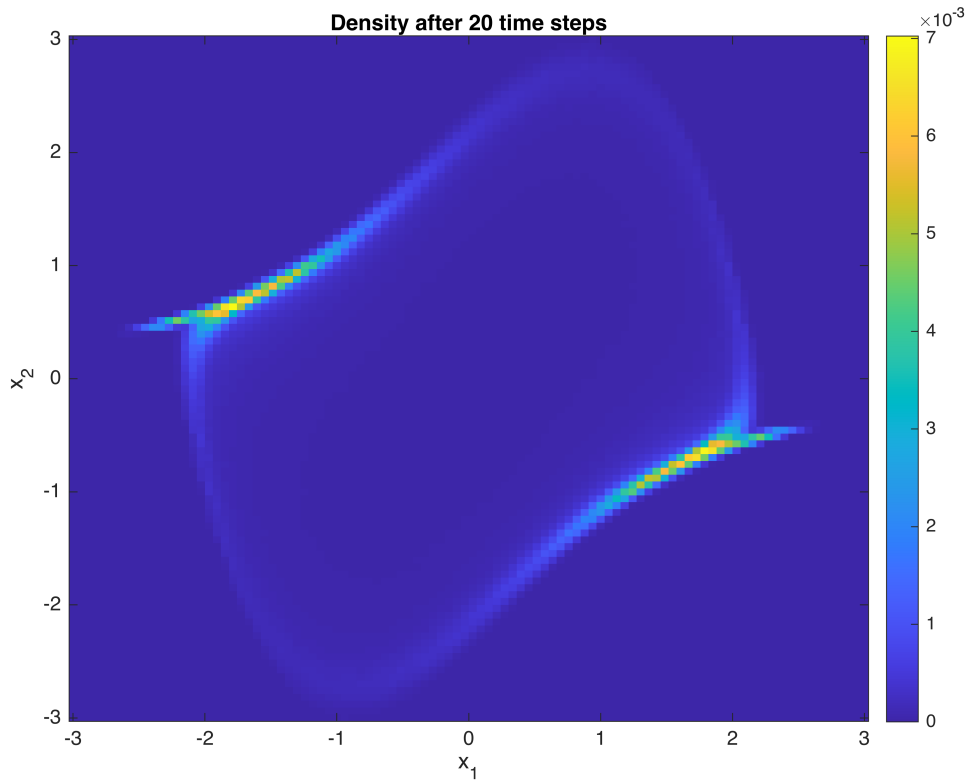
Use the Monte Carlo approach to obtain the discretization $\overline{P}$ of the - P_t of the Van der Pol system with $t = 0.1$ on the domain $[-3, 3] \times [-3, 3]$ that is partitioned via a 100×100 grid. Illustrate the evolution of the state density when initialized with a uniform density with height 1. Also examine the eigenvalue problem for the eigenvalue 1 and illustrate the resulting discretization of the invariant density.

```
% Setting up the domain
x1_range = linspace(-3, 3, 100);
x2_range = linspace(-3, 3, 100);
t_span = [0 0.1];

% Initializing the matrices to store endpoints and transition probabilities
endpoints = zeros(100, 100, 2);

% Monte Carlo simulation sampling number
points_per_cell = 1000;

% Initialize matrices to store endpoints and transition probabilities
transition_matrix = zeros(100*100, 100*100);

% Defining the size of each cell in the grid
dx1 = (max(x1_range) - min(x1_range)) / (length(x1_range) - 1);
dx2 = (max(x2_range) - min(x2_range)) / (length(x2_range) - 1);

% Solve the system for each grid point and populate the transition matrix
for i = 1:length(x1_range)
    for j = 1:length(x2_range)
        % Defining the edges of the cell
        x1_edges = [x1_range(i) - dx1/2, x1_range(i) + dx1/2];
        x2_edges = [x2_range(j) - dx2/2, x2_range(j) + dx2/2];

        % Ensure that the edges are within the domain limits
```

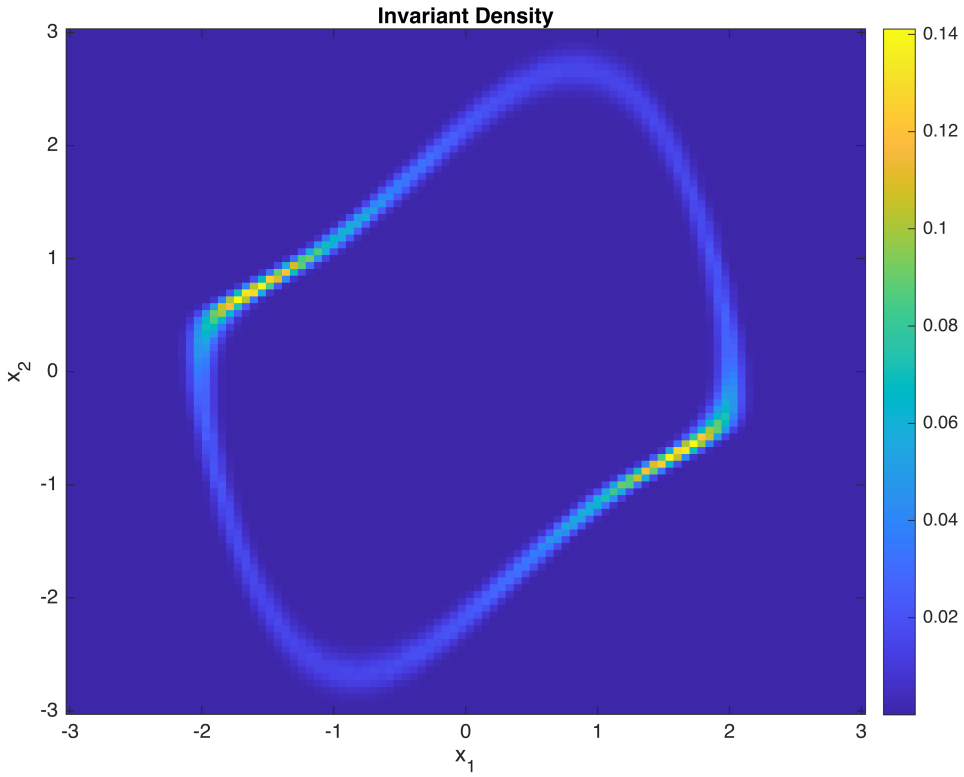
```
        x1_edges = max(min(x1_edges, max(x1_range)), min(x1_range));
        x2_edges = max(min(x2_edges, max(x2_range)), min(x2_range));

        % Generate points uniformly distributed within the cell
        cell_points_x1 = rand(points_per_cell, 1) * (x1_edges(2) -
x1_edges(1)) + x1_edges(1);
        cell_points_x2 = rand(points_per_cell, 1) * (x2_edges(2) -
x2_edges(1)) + x2_edges(1);

        for k = 1:points_per_cell
            % Initial condition for each point
            x0 = [cell_points_x1(k); cell_points_x2(k)];

            % Solve the Van der Pol system for each point
            [T, X] = ode45(@vanDerPolSystem, t_span, x0);

            % Map the endpoint to the closest grid point
            [~, ix] = min(abs(x1_range - X(end, 1)));
            [~, iy] = min(abs(x2_range - X(end, 2)));
            index_end = sub2ind([100, 100], ix, iy);
            index_start = sub2ind([100, 100], i, j);

            % Updating the transition matrix
            transition_matrix(index_end, index_start) =
transition_matrix(index_end, index_start) + 1;
        end
    end
end

% Normalize the transition matrix so each column sums to 1
transition_matrix = transition_matrix ./ sum(transition_matrix, 1);

% Define the initial density to be a uniform distribution
initial_density = ones(100, 100) / (100 * 100);
initial_density_vector = reshape(initial_density, [100*100, 1]);

% Displaying the initial density
figure;
imagesc(x1_range, x2_range, initial_density);
title('Initial Uniform Density');
xlabel('x_1');
ylabel('x_2');
colorbar;
set(gca, 'YDir','normal');
```




```
% Number of time steps to evolve
num_steps = 20;

% Evolve the density using the transition matrix
evolved_density_vector = initial_density_vector;
for step = 1:num_steps
    evolved_density_vector = transition_matrix * evolved_density_vector;
end

% Reshape the evolved density vector back to a 100x100 grid
evolved_density = reshape(evolved_density_vector, [100, 100]);

% Displayig the evolved density after num_steps
figure()
imagesc(x1_range, x2_range, evolved_density');
title(['Density after ', num2str(num_steps), ' time steps']);
xlabel('x_1');
ylabel('x_2');
colorbar;
set(gca, 'YDir','normal');
```




```
% Converting the transition matrix to a sparse format
transition_matrix_sparse = sparse(transition_matrix);

% Finding the dominant eigenvalue and eigenvector and requesting the
largest eigenvalue
opts.isreal = true; % since our matrix is real
[V_sparse, D_sparse] = eigs(transition_matrix_sparse, 1, 'largestreal',
opts);

% Using the dominant eigenvector as the invariant density vector
invariant_density_vector = V_sparse(:, 1);

% Ensuring the invariant density is non-negative and normalized
invariant_density_vector = abs(invariant_density_vector);

% Reshaping the invariant density vector to a 100x100 grid
invariant_density = reshape(invariant_density_vector, [100, 100]);

% Displaying the invariant density
figure()
imagesc(x1_range, x2_range, invariant_density');
title('Invariant Density');
xlabel('x_1');
ylabel('x_2');
colorbar;
set(gca, 'YDir','normal');
```



2. The Matlab-file `snapshot_data.mat` is a data set containing 50,000 sample points in the state space as well as the evolution of these points under the flow of a nonlinear oscillatory system $\dot{x} = f(x)$ for a (small) period time. The underlying dynamical system admits a conserved quantity of the form

$$V(x_1, x_2) = Ax_1^2 + Bx_2^2 + Cx_1^3$$

i.e., for any trajectory $t \to x(t)$, it holds that $V(x(t)) \equiv$ constant.
Use a Koopman operator-based approach (Extended Dynamic Mode Decomposition) with a function library consisting of the terms $x_1, x_2, x_1^2, x_1x_2, x_2^2, x_1^3, x_1^2x_2, x_1x_2^2, x_2^3$ to computationally determine this conserved quantity using only the snapshot data.

```
% Loading in the snapshot data
load snapshot_data.mat

% X is the initial data
X = [ x ;
      x(1,:).^2 ;
      x(1,:).*x(2,:);
```

```
      x(2,:).^2 ;
      x(1,:).^3 ;
      x(1,:).^2.*x(2,:);
      x(1,:).*x(2,:).^2;
      x(2,:).^3  ];

% X2 is the next step
X2 = [ x2 ;
      x2(1,:).^2 ;
      x2(1,:).*x2(2,:);
      x2(2,:).^2 ;
      x2(1,:).^3 ;
      x2(1,:).^2.*x2(2,:);
      x2(1,:).*x2(2,:).^2;
      x2(2,:).^3  ];

% Finding the inverse
% Creating the matrix exponential
e_A = X2 * pinv(X);

% Finding the left eignevectors
[A_eig_vec, A_eig_val] = eig(e_A')
```

```
A_eig_vec = 9×9 complex
  -0.0009 - 0.0009i  -0.0009 + 0.0009i   0.0002 - 0.0007i   0.0002 + 0.0007i ···
  -0.0006 - 0.0001i  -0.0006 + 0.0001i  -0.0002 - 0.0003i  -0.0002 + 0.0003i
  -0.0060 + 0.1633i  -0.0060 - 0.1633i   0.6910 + 0.0000i   0.6910 + 0.0000i
   0.1487 + 0.0069i   0.1487 - 0.0069i  -0.0009 - 0.6794i  -0.0009 + 0.6794i
   0.0025 - 0.0361i   0.0025 + 0.0361i  -0.1705 + 0.0005i  -0.1705 - 0.0005i
   0.0031 - 0.5117i   0.0031 + 0.5117i  -0.1268 + 0.0001i  -0.1268 - 0.0001i
  -0.7421 + 0.0000i  -0.7421 + 0.0000i  -0.0013 - 0.0457i  -0.0013 + 0.0457i
   0.0004 + 0.3655i   0.0004 - 0.3655i  -0.1131 + 0.0013i  -0.1131 - 0.0013i
   0.0608 + 0.0004i   0.0608 - 0.0004i   0.0002 + 0.0293i   0.0002 - 0.0293i
A_eig_val = 9×9 complex
   0.8193 + 0.5708i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i ···
   0.0000 + 0.0000i   0.8193 - 0.5708i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.9254 + 0.3814i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.9254 - 0.3814i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
```

```
% Finiding the eigenvalue 1 to be the 5th one
A_eig_val(:,5)
```

```
ans = 9×1
         0
         0
         0
         0
    1.0000
         0
         0
         0
```

```
0
```

```
% Checking why of the components of the eigen vector make significant
% contributions
A_eig_vec(:,5)
```

```
ans = 9×1
   -0.0000
    0.0000
   -0.9577
    0.0000
   -0.2394
    0.1596
   -0.0000
   -0.0000
   -0.0000
```

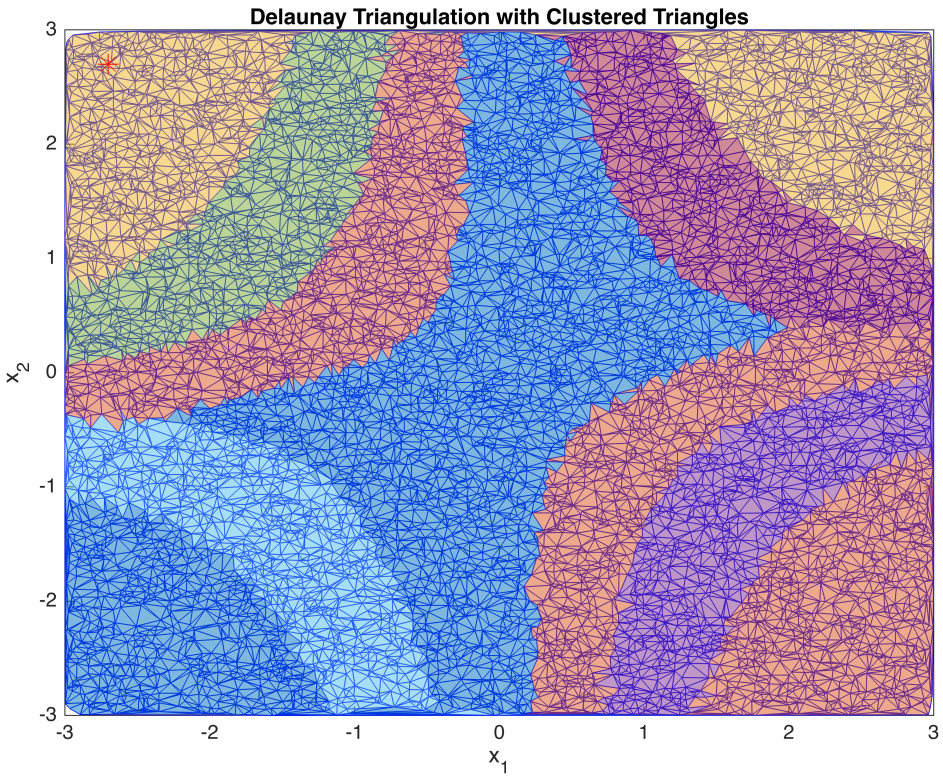
The resulting equation is as follows:

$$V(x_1, x_2) = -0.9577x_1^2 - 0.2394x_2^2 + 0.1596x_1^3$$

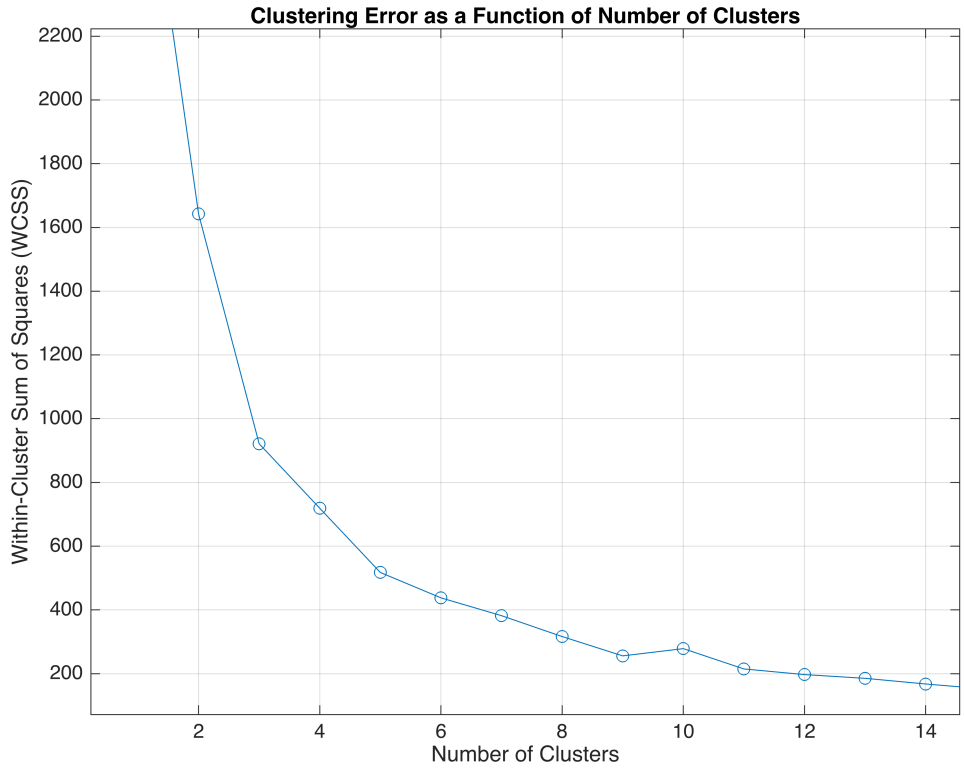
3. Visualize the results of clustering a state space based on the affine approximations of the flow for the Van der Pol oscillator and plot the clustering error as a function of the number of clusters.

```
% This is the data file containing triangulation from homework 1
load('HW1data.mat');
[idx, C] = kmeans(translation, 10, 'MaxIter',1000,'Distance', 'cosine');

% Plot original triangulation
figure();
triplot(DT);
xlim([-3, 3]);
ylim([-3, 3]);
hold on;

% Highlight all triangles based on their cluster
colors = lines(20);


% Loop over each triangle to color it based on its cluster
for i = 1:size(DT.ConnectivityList, 1)
    vertices = DT.ConnectivityList(i, :);
    triPoints = DT.Points(vertices, :);
    clusterIdx = idx(i);
    patch('Faces', [1 2 3], 'Vertices', triPoints, 'FaceColor',
colors(clusterIdx, :), 'FaceAlpha', 0.5, 'EdgeColor', 'none');
end

% Marking a specific point
p = [-2.7, 2.7];
plot(p(1), p(2), 'r*', 'MarkerSize', 10);
```

```
title('Delaunay Triangulation with Clustered Triangles');
xlabel('x_1'); ylabel('x_2');
hold off;
```




```
% Number of clusters
minClusters = 1;
maxClusters = 20;
numClusters = minClusters:maxClusters;

wcss = zeros(length(numClusters), 1);

for k = numClusters
    [idx, C, sumd] = kmeans(translation, k, 'MaxIter',1000, 'Distance',
'cosine');
    wcss(k-minClusters+1) = sum(sumd);
end

% Plotting the clustering error (WCSS)
figure;
plot(numClusters, wcss, '-o');
title('Clustering Error as a Function of Number of Clusters');
xlabel('Number of Clusters');
ylabel('Within-Cluster Sum of Squares (WCSS)');
grid on;
```



From the plot, it can be seen that there is diminishing gain in preformance as the number of clusters increase past 8. 8 cluster seems to be optimal.

```
function dx = vanDerPolSystem(t, x)
    % Van der Pol system equations
    dx = zeros(2,1);
    dx(1) = x(2);
    dx(2) = (1 - x(1)^2) * x(2) - x(1);
end
```