Takens Embeddings

Issar Amro

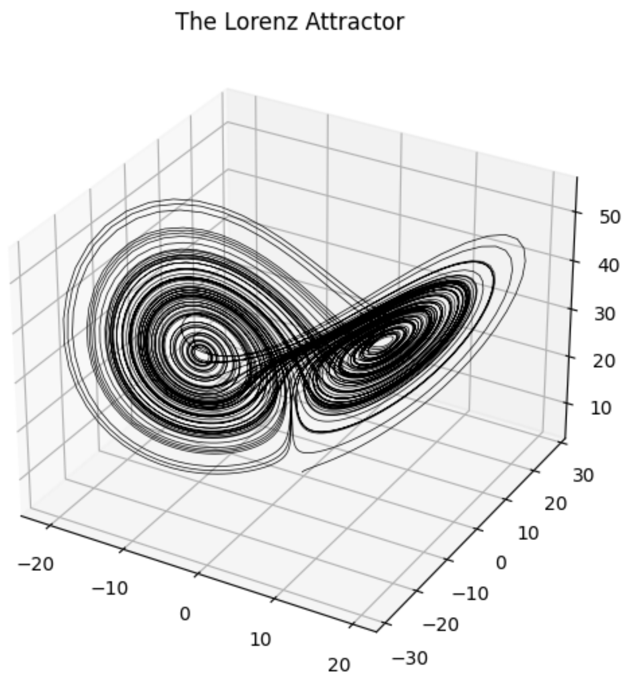
Setup.

Given a dynamical object $\subset \mathbb{R}^n$ (n-dimensional Euclidean space) which is the solution of a system of n coupled differential equations, if we project the object into a lower dimensional space $\mathbb{R}^{n-m}$ for $0 < m < n$, we get m time series solutions. Having only these solutions and no other information about the other solutions or the governing equations, can we recover the dynamics of the original n-dimensional object?
An example of such an object is the famous Lorenz attractor which is the solution to this system

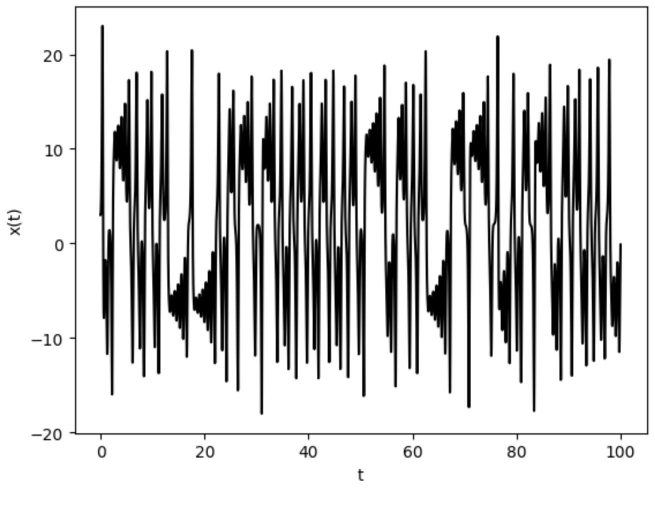
$$\begin{cases} \dot{x} = \sigma(y - x) \\ \dot{y} = rx - y - xz \\ \dot{z} = xy - bz \end{cases}$$

Solving the system with these parameters (σ = 10, b = 8/3, r = 28), using RK4 scheme, and plotting the three-dimensional solution with the added noise in the phase/solution space (x(t), y(t), z(t)), we get

```
def dx(x, y, sigma):
    return sigma*(y - x)

def dy(x, y, z, r):
    return r*x - y - x*z

def dz(x, y, z, b):
    return x*y - b*z

def RK4_Lorenz(x, y, z, dx, dy, dz, sigma, r, b, h):
    for i in range(0, len(x) - 1):
        k1x = h*dx(x[i], y[i], sigma)
        k2x = h*dx(x[i] + 0.5*k1x, y[i], sigma)
        k3x = h*dx(x[i] + 0.5*k2x, y[i], sigma)
        k4x = h*dx(x[i] + k3x, y[i], sigma)
        x[i+1] = x[i] + (k1x + 2*k2x + 2*k3x + k4x)/6

        k1y = h*dy(x[i], y[i], z[i], r)
        k2y = h*dy(x[i], y[i] + 0.5*k1y, z[i], r)
        k3y = h*dy(x[i], y[i] + 0.5*k2y, z[i], r)
        k4y = h*dy(x[i], y[i] + k3y, z[i], r)
        y[i+1] = y[i] + (k1y + 2*k2y + 2*k3y + k4y)/6

        k1z = h*dz(x[i], y[i], z[i], b)
        k2z = h*dz(x[i], y[i], z[i] + 0.5*k1z, b)
        k3z = h*dz(x[i], y[i], z[i] + 0.5*k2z, b)
        k4z = h*dz(x[i], y[i], z[i] + k3z, b)
        z[i+1] = z[i] + (k1z + 2*k2z + 2*k3z + k4z)/6
    return x, y, z

h = 0.01
t = np.arange(0, 100, h)
x = np.zeros(len(t))
y = np.zeros(len(t))
z = np.zeros(len(t))
x[0] = 1
y[0] = 1
z[0] = 1
sigma = 10
b = 8/3
r = 28
xs, ys, zs = RK4_Lorenz(x, y, z, dx, dy, dz, sigma, r, b, h)
xs = xs + 0.5*np.random.randint(-10, 10)
ys = ys + 0.5*np.random.randint(-10, 10)
zs = zs + 0.5*np.random.randint(-10, 10)
```



Now let us consider only the $x(t)$ noisy solution and assume we do not know the generating system of equations and the dimensionality of the attractor. Is it possible to reconstruct the attractor based on the information presented by the one-dimensional array solution in such a way that preserves the properties of the original, unknown, dynamical system?



Takens Theorem.

For a compact C^2 manifold M of dimension d and a generic pair of C^2 diffeomorphism T: $M \to M$ and C^2 observable x: $M \to \mathbb{R}$, the k-delay coordinate map for x is a C^2 embedding provided that k

$\geq$ 2d + 1. The k-delay coordinate map $\mathbb{R} \to \mathbb{R}^k$ is defined as

$$x(t) \to (x(t), x(t-\tau), x(t-2\tau), ..., x(t-(k-1)\tau))$$

Where τ is the time delay such that $\tau > 0$.

Definitions.

Manifold: a generalization and abstraction of the notion of a curved surface; a manifold is a topological space that is modeled closely on Euclidean space locally but may vary widely in global properties.
Diffeomorphism: bijective, reversible, and both ways differentiable map between 2 smooth manifolds. A diffeomorphism is a topology-preserving transformation, and many dynamical quantities are preserved under such transformations.

Time Delay.

Takens theorem is usually used to reconstruct strange attractors out of experimental data, for which there is noise. As such, the choice of delay time becomes important. Whereas for data without noise, any choice of strictly positive delay time value is valid, for noisy data, if the value of τ was arbitrary, we might encounter overfolding of the reconstructed attractor which makes the process of extracting the dynamics more complicated. Furthermore, for such "incorrect" values of τ, we might get false information out of the reconstructed attractor for, even though mathematically the choice of τ should not affect the dynamics, computational limitation introduces errors when presented by over-complicated reconstructions.

A method to find the correct time delay value is proposed by Fraser and Swinney by treating the coordinate map as a basis for the reconstructed m-dimensional space. Ideally, a basis has to have orthogonal components but since the data we have are noisy (assuming they are data collected by experiment), we cannot always guarantee finding an orthogonal set of m elements. The alternative is to find the value of the time delay τ that minimizes the general dependency of the set elements on each other. Fraser and Swinney introduced "mutual information" I, a parameter whose first minimum is associated with the "right" time delay τ. I measures the general (rather than only the linear) dependence of 2 variables, hence it provides a better criterion for the choice of τ than the prior proposed method of the autocorrelation function whose zero is associated with the right time delay since the system we are dealing with is nonlinear. But why the first minimum? Again, to avoid overfolding and hence avoid computational complications, noting that sometimes in literature they take the second minimum or the first maximum of I, all depending on the system and the context/method. For our context, the first minimum works.

Mutual information has the definition of entropy, i.e. the average amount of information gained from a measurement of a variable. Taking $P(x_i)$ as the probability density at x_i, the entropy of the variable x would be expressed as

$$S(x) = -\sum_{i} P(x_i) ln(P(x_i))$$

We are interested in measuring how dependent the values of x(t + τ) and x(t) are, call them the coupled system (X, Y).
To find I, we first ask given that X has been measured and is found to be X_i, what uncertainty is there in the measurement of Y? The answer is given by the conditional entropy

$$S(Y|x_i) = -\sum_{i} P_{y|x}(y_i|x_i) ln(P_{y|x}(y_i|x_i))$$

$$= -\sum_{i} \left[\frac{P_{xy}(x_i, y_i)}{P_x(x_i)} \times ln\left(\frac{P_{xy}(x_i, y_i)}{P_x(x_i)}\right) \right]$$

Where $P_{y|x}(y_i|x_i)$ is the probability that a measurement of y will yield y_i and $P_{xy}(x_i, y_i)$ is the probability of the intersection of the 2 measurements. The second question is, given that x has been measured at t, what is the average uncertainty in a measurement of x at (t + τ)? The answer is given by averaging $S(Y|x_i)$ over x_i.

$$S(Y|X) = \sum_{i} P_x(x_i) S(Y|x_i) = -\sum_{i,j} P_{xy}(x_i, y_i) ln\left(\frac{P_{xy}(x_i, y_i))}{P_x(x_i)}\right) = S(X, Y) - S(X)$$

But

$$S(X, Y) = -\sum_{i,i} P_{xy}(x_i, y_i) ln(P_{xy}(x_i, y_i))$$

S(Y) is the uncertainty in y without accounting for X and $S(Y|X)$ is the uncertainty of Y given a measurement in X. Hence, the amount that a measurement of X reduces the uncertainty in Y is defined as the mutual information between X and Y

$$I(X, Y) = S(Y) - S(Y|X) = S(Y) + S(X) - S(X, Y)$$

Now, to find a Python algorithm that calculates the mutual information for 2 given variables, we note that the probability distribution of a variable is given by the histogram of the distribution of all the measurements of this variable. We also note that the used probability distributions should be normalized.

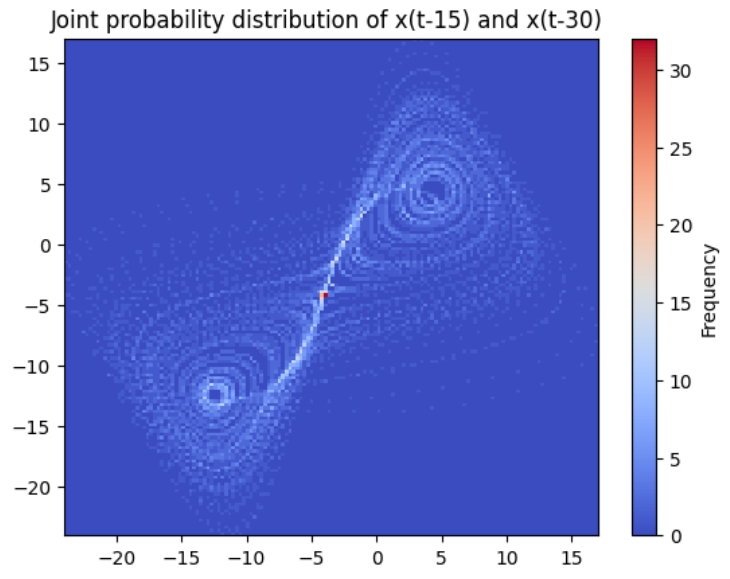
I first implemented a function that returns the entropy given an unnormalized probability distribution p

```
def entropy(p):
    p_norm = p / np.sum(p)
    p_norm = p_norm[np.nonzero(p_norm)]
    S = -np.sum(p_norm* np.log(p_norm))
    return S
```

Then, I implemented a function that calculates the probability distributions of 2 given variables and then uses the entropy function to return the mutual information of the 2 variables. The first output of Numpy histogram function is an array containing the frequencies of data points in each bin, i.e. the distribution of the data in the bins.

```
def MI(X,Y,bins):
    p_XY = np.histogram2d(X,Y,bins)[0]
    p_X = np.histogram(X,bins)[0]
    p_Y = np.histogram(Y,bins)[0]
    S_X = entropy(p_X)
    S_Y = entropy(p_Y)
    S_XY = entropy(p_XY)
    MI = S_X + S_Y - S_XY
    return MI
```

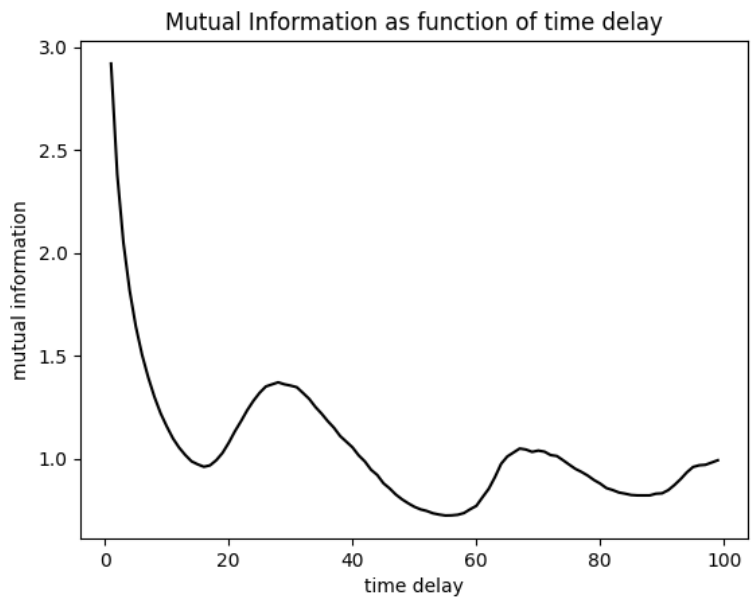
We need to use the 2 dimensional probability distribution which is defined as the 2 dimensional histogram, we can visualize an example of such distribution using a heat map



Accounting for all the elements of the solution array (x), and taking the subarrays v_1 and v_2 to be 2 consecutive time-delayed coordinates to be potentially used for the reconstruction. I looped over an interval [1, 100] of potential time delay values and found the mutual information value of (v_1, v_2) for each time delay value.

```
MI_vals = []
for i in range(1,100):
    v1 = xs[0:len(xs) - i]
    v2 = xs[i : len(xs)]
    MI_vals.append(MI(v1, v2, 1000))
```

Plotting the mutual information values as function of the time delay values, we get



Now, finding the minima of this graph and storing the corresponding τ values,

```
mins = []
for i in range(1, len(MI_vals)-1):
    if MI_vals[i] < MI_vals[i-1] and MI_vals[i] < MI_vals[i+1]:
        mins.append(i)
```

we get that the value of τ corresponding to the first minimum of the mutual information is $\tau = 15$. So we shall use this value of τ for this reconstruction problem.

Dimensionality.

Having found the right time delay, it remains now to decide how many bases we need from (x(t), x(t-τ), x(t-2τ), x(t-3τ)...) to reconstruct the original system. Again, assuming we do not have any information other than the x solution about our system. Takens theorem posed a lower bound on the chosen embedding dimension which is 2d + 1 where d is the dimensionality of the original attractor. This condition is stronger than needed and might over-complicate the problem causing overfolding in the embedded attractor which would lead to false extracted information.

False Nearest Neighbors.

We want to find the smallest dimensionality that ensures there are no intersections in the orbits of the reconstructed attractor since intersections would violate the uniqueness of the solutions to the system. If we don't allow enough dimensions, we will encounter such crossings.
Kennel et al. suggested identifying crossings for a given dimension by detecting close neighboring points and studying their separation when the dimension is increased. If the separation increases with the dimension, this means that these neighboring points are "false neighbors".

Checking the points in the given solution array and identifying neighboring points whose Euclidean separation is less than some threshold ϵ such that, for $j \neq i$,

$$\epsilon - |x_i - x_j| > 0$$

We find the square of the Euclidean distance between them, in a given dimension d, which is defined as

$$R_d^2 = \sum_{k=0}^{d-1} \left| x(t + k\tau) - x'(t + k\tau) \right|^2$$

where x′ is the nearest point (up to ϵ) to x. If we increase the dimension by 1, we get a new term in the delayed coordinates. We account for that by adding the new term to the distance getting

$$R_{d+1}^2 = R_d^2 + \left| x(t + d\tau) - x'(t + d\tau) \right|^2$$

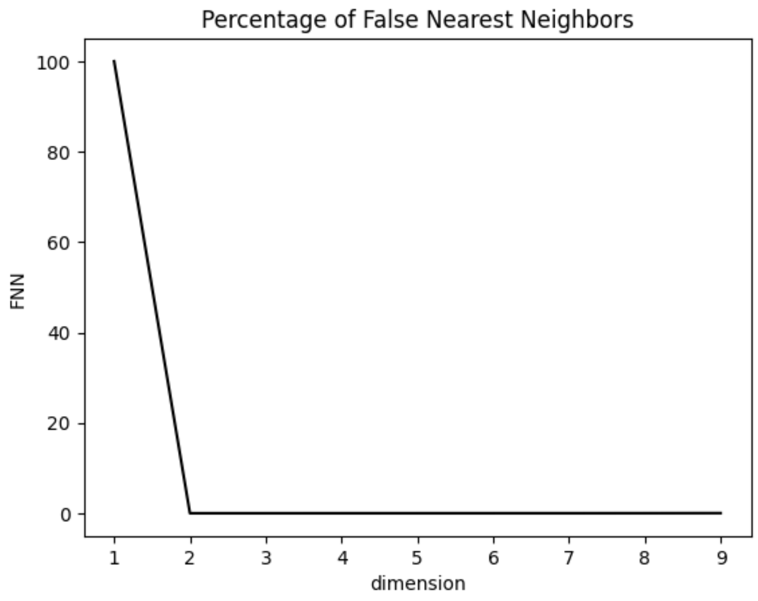
So we are tracking how the distance between these specific points is changing as we increase the dimensionality. For each iteration over the dimensions, we find the relative distance and check how much the increase in dimension has changed the distance between our points, such that if the relative distance between the 2 measured distances is greater than some tolerance, then our points were false neighbors in the lower dimension

$$\sqrt{\frac{R_{d+1}^2 - R_d^2}{R_{d+1}^2}} > \text{tol}$$

To code this, I looped over values of dimensions going from 1 to 9, and at each iteration, I looped over the indices of the time series solution checking for nearest neighbors where I took the criterion to be $\epsilon = 0.5$. When 2 neighboring points are found, I find their Euclidean separation up to the corresponding iteration (dimension) and then increase the dimension by 1 to find the new distance and then the relative distance (between the distance in dimension d and the distance in dimension d+1). I took the criterion for the relative distance to be $tol = 5$. If after the increase in dimension, the relative distance exceeds the proposed threshold, I increment the number of false neighbors and store the ratio of false to all points (the portion of the points that represents the false nearest neighbors) corresponding to the dimension.

```
eps = 0.5
tau = 15
n = len(xs)
arr = []
for o in range(1, 10):
    di = 1
    d = 1
    for i in range(len(xs) - (o * tau) - 1):
        for j in range(len(xs) - (o * tau) - 1):
            if j != i:
                if (eps - np.abs(xs[i] - xs[j])) > 0:
                  R1 = 0
                  for k in range(1, o):
                     R1 = R1 + (xs[i + (k - 1) * tau] - xs[j + (k - 1) * tau])**2
                  R2 = R1 + (xs[i + o * tau] - xs[j + o * tau])**2
                  if R1 != 0 and np.sqrt((R2 - R1)/ R1) > 5:
                     di = di +1
                  d = d + 1
    arr.append(d/di)
```

Plotting the percentage of False Nearest neighbors for the different dimensions, we get

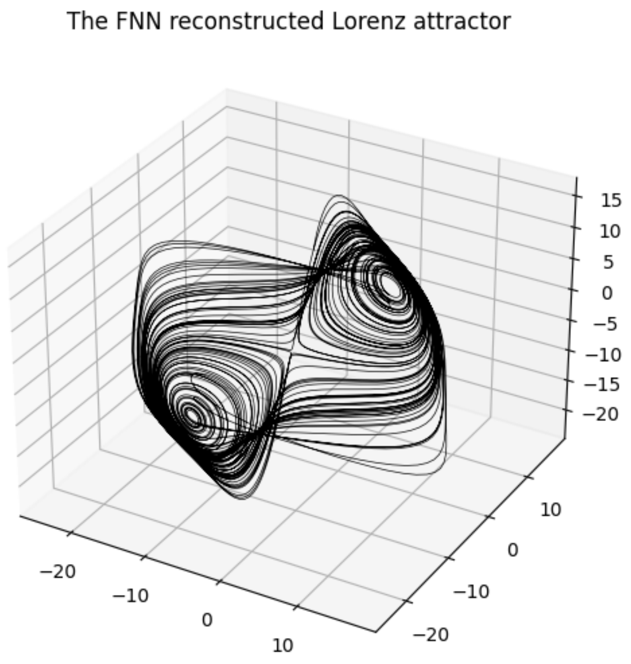


The percentage of the false neighbors drops dramatically around the dimension value 2 and converges to zero after 2. We cannot represent the delayed coordinates in dimension ”around 2”, so we take the ceiling, being 3, as the embedding dimension that is just enough to represent the reconstructed attractor. We notice how, taking larger values than 3 also guarantees having 0

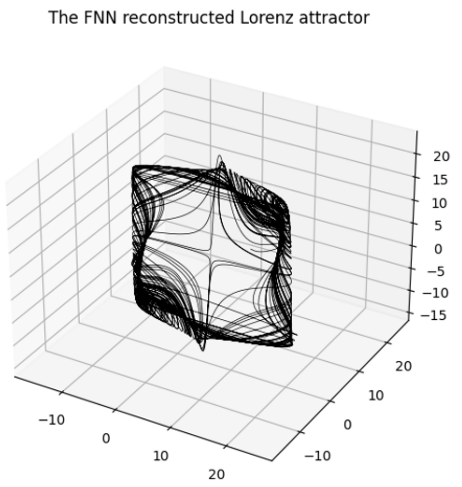
intersections which is why Takens theorem works.
Plotting the first 3 delayed versions of $x(t)$ corresponding to $\tau = 15$

$$(v_1, v_2, v_3) = (x(t),\ x(t-\tau),\ x(t-2\tau))$$

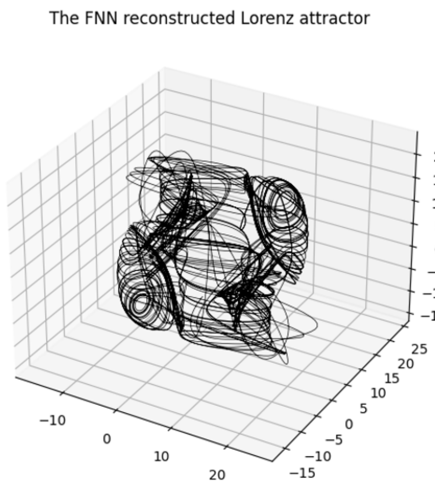
```
v1 = xs[0: len(xs) - 30]
v2 = xs[15: len(xs) - 15]
v3 = xs[30: len(xs)]
```



This reconstructed object does not have the same geometry as the Lorenz, but it does resemble it by having 3 fixed points and oscillations about 2. However, this object, according to Takens theorem, is an embedding that is topologically equivalent to the Lorenz system i.e. it contains the same information/dynamics of the Lorenz system
I mentioned before that sometimes in the literature they deal with attractors constructed by taking the first maximum or the second minimum of the mutual information instead of the first minimum. Out of curiosity, I found the attractors for the first maximum $\tau = 27$ and the second minimum $\tau = 54$ and got



$\tau = 27$



$\tau = 54$

We can see how these attractors look more deformed than the one with $\tau = 15$. Even though they carry the same dynamics, they are more complicated to study computationally.

SVD Decomposition.

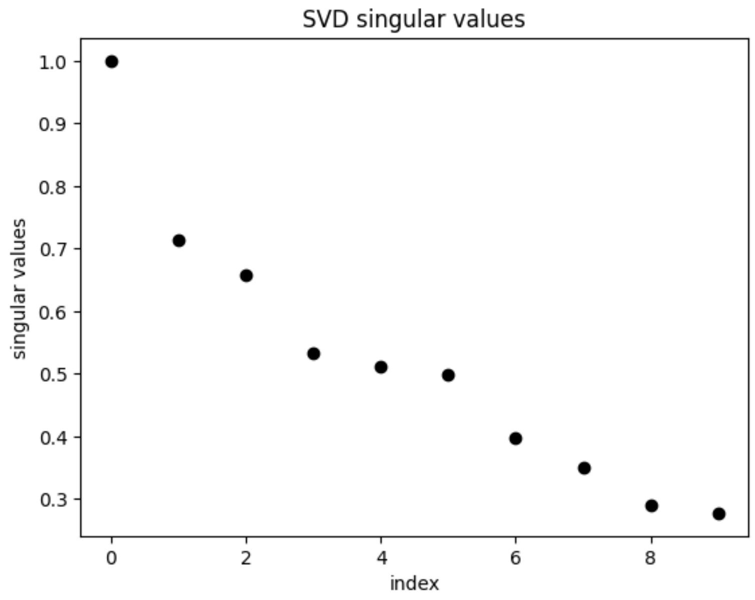
Another way to find the correct dimensionality of the embedding is by stacking the data we have from our variable $x(t)$ in a matrix where each column corresponds to a time-delayed version of the measurements, we get

$$H = \begin{pmatrix} x(t_1) & x(t_1-\tau) & x(t_1-2\tau) & ... & x(t_1-q\tau) \\ x(t_2) & x(t_2-\tau) & x(t_2-2\tau) & ... & x(t_2-q\tau) \\ x(t_3) & x(t_3-\tau) & x(t_3-2\tau) & ... & x(t_3-q\tau) \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x(t_p) & x(t_p-\tau) & x(t_p-2\tau) & ... & x(t_p-q\tau) \end{pmatrix}$$

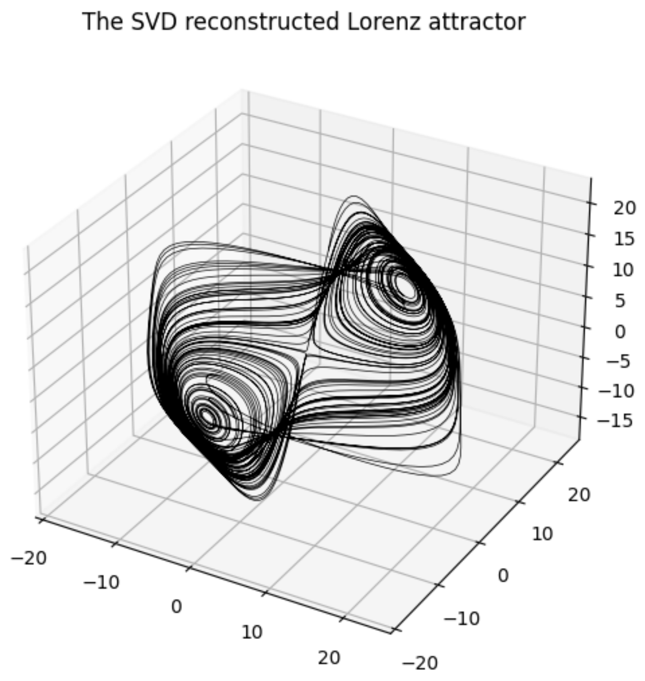
Performing singular value decomposition on H, we store the singular values, which represent the importance or strength of each mode. A mode, in our case, is a temporal pattern/delay.

```
tau = 15
H = np.zeros((10,len(xs)-9*tau-1))
for i in range(0, 10):
    H[i,:] = xs[tau*i: len(xs)-9*tau-1 + i*tau]
H = H.T
U, S, V= np.linalg.svd(H)
```

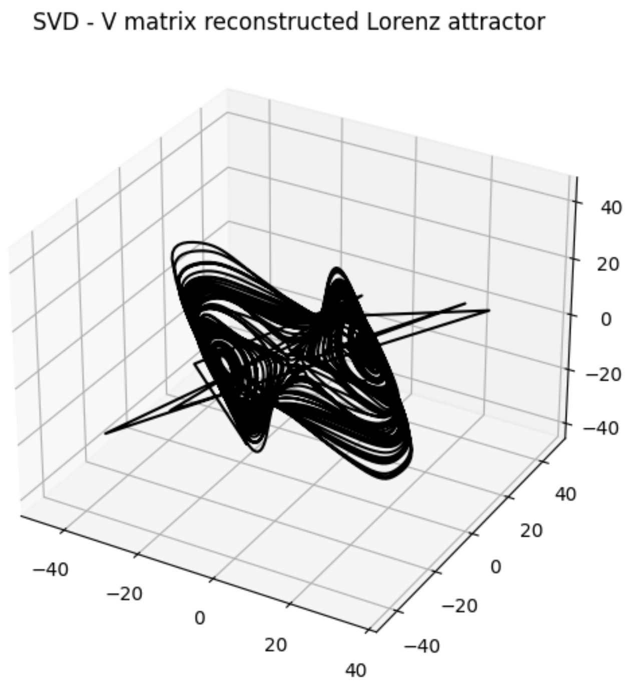
Plotting the normalized singular values for $\tau = 27$ which corresponds to the second minimum in the mutual information since it gave me clearer results on the dominants singular values, we get



We notice how the first 3 singular values are the dominant ones, which gives us an idea on the enough dimensions needed to reconstruct the dynamics. Taking the first 3 columns of our H matrix, which represent 3 arrays of delayed versions of the measurements x(t), and plotting them against each other ($H[:, 0], H[:, 1], H[:, 2]$), we recover the embedded attractor we got using the mutual information method



We can also consider the V matrix, which is the orthogonal matrix that contains all the temporal information of the system. Taking the first 3 columns of V scaled by the corresponding singular values ($V[:,0] * S[0],\ V[:,1] * S[1],\ V[:,2] * S[2]$) and plotting them



We notice that this attractor is noisy, something we would expect by ignoring the other singular values.

Maximal Lyapunov Exponent.

Having constructed 3 objects that I claimed are topologically equivalent to the Lorenz attractor and contain the same dynamics, how can I make sure that this is the case? I choose a dynamical variable and calculate it for the Lorenz system and for the 3 reconstructed attractors and compare. I attempted doing this for the maximal Lyapunov exponent.
The Lyapunov exponent is a measure of the stretching/expansion in the flow in the phase (solution) space, and the maximal Lyapunov exponent attained by tracing 2 points in the phase space and recording their Euclidean distance at 2 instances in time is a measure of how chaotic the studied system is and can be found from this relation $|\delta(t_f)| = |\delta_0| e^{\lambda t_f}$ getting

$$\lambda = \frac{1}{t_f} ln\left(\frac{|\delta(t_f)|}{|\delta_0|}\right)$$

For the original Lorenz system, knowing the generating equations, to find the Lyapunov exponent we solve the system for 2 initial conditions and compare their Euclidean distance at time = 0 ($\delta(0)$) to their Euclidean distance after allowing them to evolve until t = t_{final} $\delta(t)$

```
h = 0.01
t = np.arange(0, 100, h)
x1 = np.zeros(len(t))
x2 = np.zeros(len(t))
y1 = np.zeros(len(t))
y2 = np.zeros(len(t))
z1 = np.zeros(len(t))
z2 = np.zeros(len(t))
sigma = 10
b = 8/3
r = 28
delta0 = [0.5, 0.2, 0.3]
x1[0] = 1
x2[0] = 1 + delta0[0]
y1[0] = 1
y2[0] = 1 + delta0[1]
z1[0] = 1
z2[0] = 1 + delta0[2]
x1s, y1s, z1s = RK4_Lorenz(x1, y1, z1, dx, dy, dz, sigma, r, b, h)
x2s, y2s, z2s = RK4_Lorenz(x2, y2, z2, dx, dy, dz, sigma, r, b, h)
delta = np.array([x2[-1] - x1[- 1], y2[- 1] - y1[- 1], z2[- 1] - z1[- 1]])
lyapunov = (1/t[- 1]) * np.log(np.linalg.norm(delta) / np.linalg.norm(delta0))
```

We get $\lambda = 0.02636$.

For the reconstructed attractors, however, we worked on them under the assumption that we know nothing about the system including the generating equations hence the approach we used on the Lorenz system cannot be applied here. What I did instead was taking the 3 embedding coordinate arrays and shifting them by the same value, say 18, and finding their Euclidean distance before and after the shifting. The distance before the shifting plays the role of $\delta(0)$ and the distance after the shifting plays the role of $\delta(t)$ for t being 18 for this example.

```
v11 = v1[0: len(v1)-18]
v21 = v2[0: len(v1)-18]
v31 = v3[0: len(v1)-18]
v12 = v1[18: len(v1)]
v22 = v2[18: len(v1)]
v32 = v3[18: len(v1)]
v0 = np.zeros(3)
v0[0] = v12[0] - v11[0]
v0[1] = v22[0] - v21[0]
v0[2] = v32[0] - v31[0]
vt = np.zeros(3)
vt[0] = v12[18] - v11[18]
vt[1] = v22[18] - v21[18]
vt[2] = v32[18] - v31[18]
lyapunov = (1/18) * np.log(np.linalg.norm(vt) / np.linalg.norm(v0))
```

• For the False Nearest Neighbors reconstructed system, I got $\lambda = 0.0258$.
• For the SVD reconstructed using the H matrix, I got $\lambda = 0.0258$, which is expected since it should be exactly the same as the FNN reconstructed one.
• For the SVD reconstructed system using the V matrix, I got $\lambda = 0.02696$.
We notice how these values are very close to the value we extracted form the Lorenz system, meaning these reconstructed attractors are indeed topologically equivalent to the original system and hence good embeddings of the original dyanamics.

Notes On Class Presentations.

Fahed: The quest to a third isolating integral of motion.

Hénon and Heiles argued that the motion around the galactic center gives rise to a 6-dimensional phase space and hence the system possesses 5 integrals of motion (isolating and non-isolating). Total energy and angular momentum are the isolating integrals of motion for such a system, but observational data suggested the existence of a third isolating integral of motion. To check for its existence required finding the Poincare map of the Henon Heiles potential. Fahed found the governing equations of motion from the potential and solved them for different initial conditions using RK4 method and plotted the map. The results were that the third isolating integral of motion exists but only for some initial conditions, so its existence is dependent on how the system starts.

Roua: communicability in brain Networks.

To find the communicability between Neurons in the brain, Roua used a Network model. Defining neurons as the nodes of such network and synapses as the connections/edges between these nodes while taking into consideration that neurons can be deactivated and can get disconnected from the network. Connectivity in a network is dependent on the number of allowed shortest paths and increases with them.
Now, to construct the communicability matrix of a brain model, Roua used the Lorenz system with the (x, y, z) time series representing the evolution of 3 neurons. Calculating the mutual information between each possible pair, she found how each pair depends on each other and constructed a 3x3 matrix, the communicability/adjacency matrix. From the adjacency matrix, one could construct a network that describes the connections between these 3 neurons. In this case, the constructed network was a small world network since it better represents the brain network.

Khalil: A brief discourse on Ergodicity.

A stochastic approach is an approach that is dependent on previous values and governed by random evolution. Khalil modeled such a process by initiating a random walker with no bias on a restricted phase space such that once the walker hits the boundary, it reflects. If the steps followed by the walker were truly random, according to the ergodic theorem, it would fill the phase space completely after a large number of iterations, and all the possible trajectories would be indistinguishable. This could be used as a test of how efficient a random number generator is. Due to computational limitations, sometimes ”loss of randomness” is encountered and we start noticing trends in the trajectories followed by the walker, this breaks stochasticity and now the trajectories become distinguishable. Khalil also discussed an interesting method called the ”Baker's map” to transform the phase space of an ergodic system under constraints that contains forbidden regions to a phase space with equally probable visited regions which would allow the system to explore the entirety of its phase space volume.

Jad: Scalar curvature in Discrete Gravity.

Is a tennis ball curved? Jad investigated curvature using Discrete Gravity tools. Discrete gravity treats manifold as discretized cells each with a volume of the Planck constant. In GR, a shift operator is applied on a vector that shifts the vector along a parallel direction (parallel to its original direction), so the vector is transported along a tangent plane to its starting point. The analog to this in discrete gravity is shifting a vector over the elementary cells and defining curvature in terms of the spin connections. In Dr. Chamseddine's paper, they introduce ”isotropic coordinates” to represent the metric and arrive at a quantity they called ″curvature scalar″ which represents the curvature term and is analog to the Ricci scalar in GR.

Garo: Laplacian coarse-graining of complex networks.

Networks that arise from studying aspects of the world are huge in size and have a large number of interacting components which makes them hard to analyze. Garo tackled a method based on statistical mechanics to reduce the size of such complex networks by merging nodes based on some criteria. The procedure is done by calculating the pseudo Laplacian and then finding the 2 most correlated nodes at each iteration. We then combine the activities of these 2 nodes and merge them into 1 ″super node″ that has the combined connections of the 2 smaller ones.

Semaan: 1D Heat Equation.

The goal was to solve the 1-dimensional heat equation

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + \beta \frac{\partial u}{\partial x}$$

discretely using the finite difference method on the left-hand side and using Chebychev polynomials for the right-hand side. The Chebychev polynomials are generated by the change of variable $x = \cos\theta$. For evenly spaced values of θ, we get points that are not evenly spaced but rather concentrated at the edges and sparse around zero. This is useful in the cases where mitigate Runge phenomena appear, and Semaan used it to avoid such phenomena when discretizing the right-hand side.