SHEAR-INDUCED CHAOS

JENNA BENDINELLI, EDRIA LI, ARMANDO MEDINA-CALDERON, TIANYI NI, SUMMER WILLIAMS, AND YUYU ZHANG

AMATH 575, *University of Washington, Seattle, WA*

Abstract. We successfully replicated two studies from the work of Lin and Young in shear-induced chaos which involve introducing periodic kicks as well as random kicks using the Poisson distribution to generate chaos. We implemented two numerical methods with guidance of multiple related literature. We furthered the research of chaos production by exploring the effect of small perturbation of amplitude with periodic kicks; we also investigated a widening distribution of kicks, where the distribution of kicks starts tight and increases the time between kicks.

1. Introduction

1.1. Paper Introduction.

In their paper *Shear-induced Chaos*, authors Lin and Young looked to "investigate the use of shear in the production of chaos" [3]. Of the four studies completed in the paper, we focused on the first two − where periodic kicks and random kicks are introduced into the system. The Lyapunov exponents are used to show when and under what conditions chaos occurs, with positive exponents meaning that there is chaos and negative exponents indicating no chaos.

The first study of this paper introduced periodic kicks into a linear shear flow with a hyperbolic limit cycle using the system:

$$\begin{aligned} \dot{\theta} &= 1 + \sigma y, \\ \dot{y} &= -\lambda y + A \sin(2\pi\theta) \cdot \sum_{n=0}^{\infty} \delta(t - nT), \end{aligned} \tag{1}$$

Date: June 7^{th} 2023.

where σ = amount of shear, λ =damping, A = amplitude of kicks, and T = time between kicks.

We use (1) to solve for the flow of the system, giving:

$$\begin{aligned} \theta_T &= \theta_0 + T + \frac{\sigma}{\lambda}[y_0 + A\sin(2\pi\theta_0)](1 - e^{-\lambda T}) \\ y_T &= e^{-\lambda T}[y_0 + A\sin(2\pi\theta_0)] \end{aligned} \tag{2}$$

The study found that the condition $(\sigma/\lambda)A(1-e^{-\lambda T})$ is a good indicator for when chaos occurs. For smaller T values, there is no chaos as the folding relationship is not well formed, but as T increases, the limit cycle begins to fold and we start to see oscillations between positive and negative Lyapunov exponents. This is because there is competition between transient (intervals of chaos) and sustained (constant) chaos. However, when $(\sigma/\lambda)A$ is larger (particularly around 3), the stretching is stronger, and sustained chaos dominates. Finally, shear induced chaos had previously been associated with weak damping, however under the right conditions they see chaos when damping is larger such as $\lambda = 1$ as long as the relationship $(\sigma/\lambda)A$ is maintained.

The second study with random kicks used the following system

$$\begin{aligned} \dot{\theta} &= 1 + \sigma y, \\ \dot{y} &= -\lambda y + \sin(2\pi\theta) \cdot \sum_n A_n \delta(t - T_n), \end{aligned} \tag{3}$$

where the time between kicks are random exponential variables and the kick amplitudes are uniformly distributed over the interval $[0.8A, 1.2A]$. The introduction of random kicks into the system makes this a random map, and by the standard theory of random maps, the lyapunov exponents are non-random and not path dependent when shear is non-zero. This means that we are able to perform the same chaos analysis that was performed in the periodic kick study, and apply it to the random kick study.

There are two main differences when it comes to the random kick study when compared to the periodic kick study, and they are that the oscillations seen in the periodic study have been

smoothed out, and the lyapunov exponents have a significantly faster convergence rate. The first difference can be seen on the resulting plots of the second study (discussed in the next section) which do not have the wild oscillations as seen in the first study. The smooth curves were caused by the introduction of the random kicks which have an averaging, or smoothing effect . The second difference is that positive lyapunov exponents are found for smaller values of $(\sigma/\lambda)A$ and smaller values of T, indicating a higher sensitivity to randomized kicks. This is because tight bursts of kicks allow a segment to stray from the limit cycle and for the shear to act on it long enough to develop the strange attractor.

1.2. **Supplemental Literature: Reliable and Unreliable Dynamics by Lin, Shea-Brown, Young.** The work on the study of reliability [2] defined reliability as follows: a system's reaction to a stimulus is independent of its initial state when the input is received. The results show that the distinctive geometries of random sinks and SRB measures provide a dichotomy between reliability and unreliability that is easy to recognize in simulations.

When we examine the flow generated by the model with ϵ set to zero, the perturbed flow is generally either quasi-periodic or characterized by limit cycles. We want to examine whether different distributions of the random kick affect the results, for example, Poisson kicks in study 2.

While our paper indicates that when the initial condition changes, Lyapunov exponents, which measure the exponential rates of separation of nearby trajectories, continuously decrease or exhibit periodic oscillations. We shall adjust the parameters λ and σ to see what their impact is on the Lyapunov exponents.

1.3. **Supplemental Literature: Review by Lin and Young.** A review article by Lin and Young [4] sheds more light on the general mechanism for chaos generated from periodically-kicked oscillators. The article focuses on periodic forcing in a generic dynamical system with limit cycles identical to (1), which supplement our understanding of the previous paper [3]. The limit cycle is easily observed for $\gamma = \{y = 0\}$ as $t \to \infty$ corresponding to the unforced equation. For the discrete

system (2) obtained after integrating (1), the rate of contraction with respect to the time interval is denoted as $e^{-\lambda T}$, and thus the term $\frac{\sigma}{\lambda}A(1-e^{-\lambda T})$ in the discrete system measures the tendency for a fold to develop in the kick map Ψ_T, which explains why factor $(1-e^{-\lambda T})$ does not stray too far away from 1.

The Lyapunov exponents are used for measuring orbital instability, or the speed at which nearby orbits diverge [4]. For the numerical study, the larger of the two Lyapunov exponents of the discrete kick map Ψ_T obtained from calculating the Jacobian is defined as follows:

$$\Lambda_{\max}(z) = \lim_{n\to\infty} \frac{1}{n} \log \|D\Psi_T^n(z)\| = \sup_{v\neq 0} \lim_{n\to\infty} \frac{1}{n} \log \|D\Psi_T^n(z)\cdot v\| \tag{4}$$

where the Jacobian, $D\Psi_T(\theta, y)$, is of the following form:

$$D\Psi_T(\theta,y) = \begin{pmatrix} 1+2\pi\frac{\sigma}{\lambda}A\cos 2\pi\theta(1-e^{-\lambda T}) & \frac{\sigma}{\lambda}(1-e^{-\lambda T}) \\ e^{-\lambda T}2\pi A\cos 2\pi\theta & e^{-\lambda T} \end{pmatrix} \tag{5}$$

The Jacobian defined by (5) is used in the numerical method for calculating Lyapunov exponents (see Appendices for code implementation).

The article has a proposition that given $\lambda T > 0$, and $\frac{\sigma}{\lambda}A$ sufficiently small, the limit cycle persists and the attractor is invariant. Moreover, the invariant cycle breaks as we increase $\frac{\sigma}{\lambda}A$. Due to the fact that $(\theta, y) = (\frac{1}{2}, 0)$ is a fixed point of Ψ_T, the rotational action of the Jacobian is strongest at the fixed point. The other proposition states that horseshoes are present in Ψ_T when $\frac{\sigma}{\lambda}A$ is sufficiently large. As we step further into our explorations, our results are in alignment with these propositions.

1.4. **Supplemental Literature: Early Numerical Study by Zaslavsky.** Early numerical research on conditions for strange attractors that develop in a non-linear dynamical system showed that a strange attractor can appear in a periodically perturbed system where the unperturbed

motion has a stable limit cycle [9]. The paper discussed a perturbed dissipative oscillation system very similar to (1).

By integrating both sides of the flow system, the paper obtained the following mapping for a discrete system for numerical analysis:

$$
\begin{aligned}
y_{n+1} &= e^{-\Lambda}(y_n + \epsilon \cos 2\pi x_n) \\
x_{n+1} &= x_n + \frac{\omega}{2\pi} + \frac{\alpha\omega}{2\pi\Lambda}(1 - e^{-\Lambda})y_n + \frac{K}{\Lambda}(1 - e^{-\Lambda}) \cos 2\pi x_n
\end{aligned} \tag{6}
$$

The geometric interpretation of stretching and folding was briefly touched on and can be understood intuitively. The mapping compresses along y due to factor $e^{-\gamma}$ and stretching in x does not happen in the vicinity of fixed points.

The method of analyzing stochasticity, or the development of chaotic behavior in the system, is to track local instability of trajectories, i.e. the rate of distance growth of vectors:

$$
D_n = [(x_n - x'_n)^2 + (y_n - y'_n)^2]^{\frac{1}{2}}, \tag{7}
$$

where x'_n and y'_n are trajectories with forced initial condition. This method from the early numerical exploration continues its vital importance, as this essential framework in carrying out propagation is one of our replication methods in Section 2.

1.5. **Supplemental Literature: Strange Attractors Proved by Wang and Young.** In a paper by Wang and Young [3] the first theorem discussed the creation of strange attractors from limit cycles. This theorem ensures that we can always connect and relax the system to hyperbolic limit cycles and find strange gravitational pulls, subject to some very vague requirements. However, it stays open to the possible presence of hyperbolic limit cycles in the system (as we shall see). There appears to be a strange attractor because the parameters controlling the kick are always

different. This phenomenon is known as transient chaos and has been extensively studied in the literature [1], and further proved by [7].

2. Reproduction Results

We were able to reproduce the results for both the periodic and random methods. In the replication process we iterated through 4×10^3 time points with mean time between kick intervals spanning from just above 0 to 20. We iterated the map (2) for angle, θ, and position, y, at each time step.

For each kick interval, T, the Lyapunov exponents were calculated across all time steps and averaged. A total of 10 iterations for each kick interval were performed to obtain the maximum and minimum Lyapunov exponents.

For the periodic model, the time between kicks and the amplitude of kicks was held constant. The initial θ_0 and y_0 were randomly selected with a uniform distribution between $(0, 1 - 10^{-8})$ and $(-0.1, 0.1)$ respectively.

The random model was set up similarly with a few adjustments. The map is the following:

$$\begin{aligned} \theta_T &= \theta_0 + T_n + \frac{\sigma}{\lambda} \cdot [y_0 + A_n \sin(2\pi\theta_0)] \cdot (1 - e^{-\lambda T_n}) \\ y_T &= e^{-\lambda T_n}[y_0 + A_n \sin(2\pi\theta_0)] \end{aligned} \tag{8}$$

Where A_n is a random amplitude and T_n is the random time between kicks. To determine the time between kicks, we used a random exponential function with the mean set to the specified time between kicks. The amplitude was selected using a uniform random distribution between $[0.8A, 1.2A]$ where the amplitude A is set to 0.1 across all replication models.

2.1. **Propagation Method.**

We used two approaches to calculate the Lyapunov exponents. The first approach involved a forward propagation of the map starting at two arbitrarily close points with a distance of ϵ_0. As the map is propagated forward for both points, if the distance between the

points increases significantly, then the system likely has a sensitive dependence on initial conditions, and is indicative of a chaotic system.

$$\lambda = \lim_{t\to\infty} \frac{1}{t} \ln \frac{||\epsilon_t||}{\epsilon_0} \tag{9}$$

The Lyapunov exponent calculation described by (9) was performed at each propagation step, where ϵ_0 is the initial distance between the initial points, and ϵ_t is the distance between points at each propagation step. Figure 1 illustrates results from using the propagation method to compute the Lyapunov exponents. To set up for the next time iteration we need to adjust our arbitrarily close point to be near the new point by taking that new point and adding the distance between the ending points divided by the square root of the ratio of the distances [6]. We were able to successfully apply this method to both the periodic and random models.

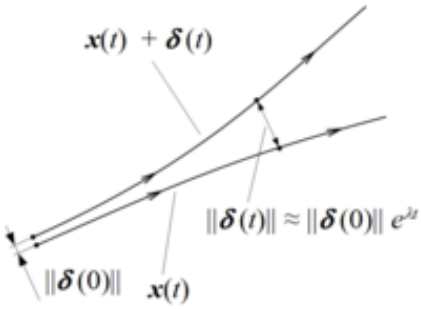

FIGURE 1. The image illustrates the concept of the Lyapunov exponent calculation using the progogation method [8]

2.2. **Jacobian Method.** The second method to compute the Lyapunov exponents involves taking the jacobian of the flow map (5). This method utilizes a linearization of the tangent vectors to determine if there is chaos. After iterating the point forward a time step we calculate the Jacobian, $D\Psi_T(\theta, y)$, at the new point.

$$\sigma_i^t(x_0) = D\Psi_T(x_0)^T D\Psi_T(x_0) \tag{10}$$

$$\lambda = \lim_{t\to\infty} \frac{1}{t} \ln \sigma_i^t(x_0) \tag{11}$$

We then take the eigenvalues of (10), and we apply (11) to calculate the Lyapunov exponents [5]. We were also able to reproduce the results of the periodic and random models with this method.

2.3. **Replication Results.** The results of both replication methods for the periodic model are in Figure 2. In the top row we have the figures from the paper [3], the middle row we have the results of the propagation method, and the bottom row has the results of the Jacobian method. The columns are based on the set of parameters used for comparison. We are able to get relatively consistent results using both models with the exception being the third column propagation method. With a slight adjustment to λ we are able to see the same pattern.

The results for the random model replication are set up in the same pattern in Figure 3. We again were able to get results relatively consistent with the paper with one exception. For the first column for the Jacobian method we need a slight adjustment to σ.

3. Novel Results

3.1. **Random Amplitude with Periodic Kicks.** We investigated combining random amplitude, A, with periodic kicks for time interval T. Similar to the distribution used in [3] for the random model, we explored uniform distribution of $[0.8A, 1.2A]$ to see if a small perturbation of A will generate chaos in either replication methods. We expect that small perturbations of A will not have significant influence on the magnitude or sign of Lyapunov exponents.

In using the propagation method, the results diverge from our initial expectations, as even a uniform distribution of range $[0.9999A, 1.0001A]$ can generate sustained chaos (see Figure 4). However, we still can observe that the general patterns of our Lyapunov exponents in the graphs match with study 1 in [3].

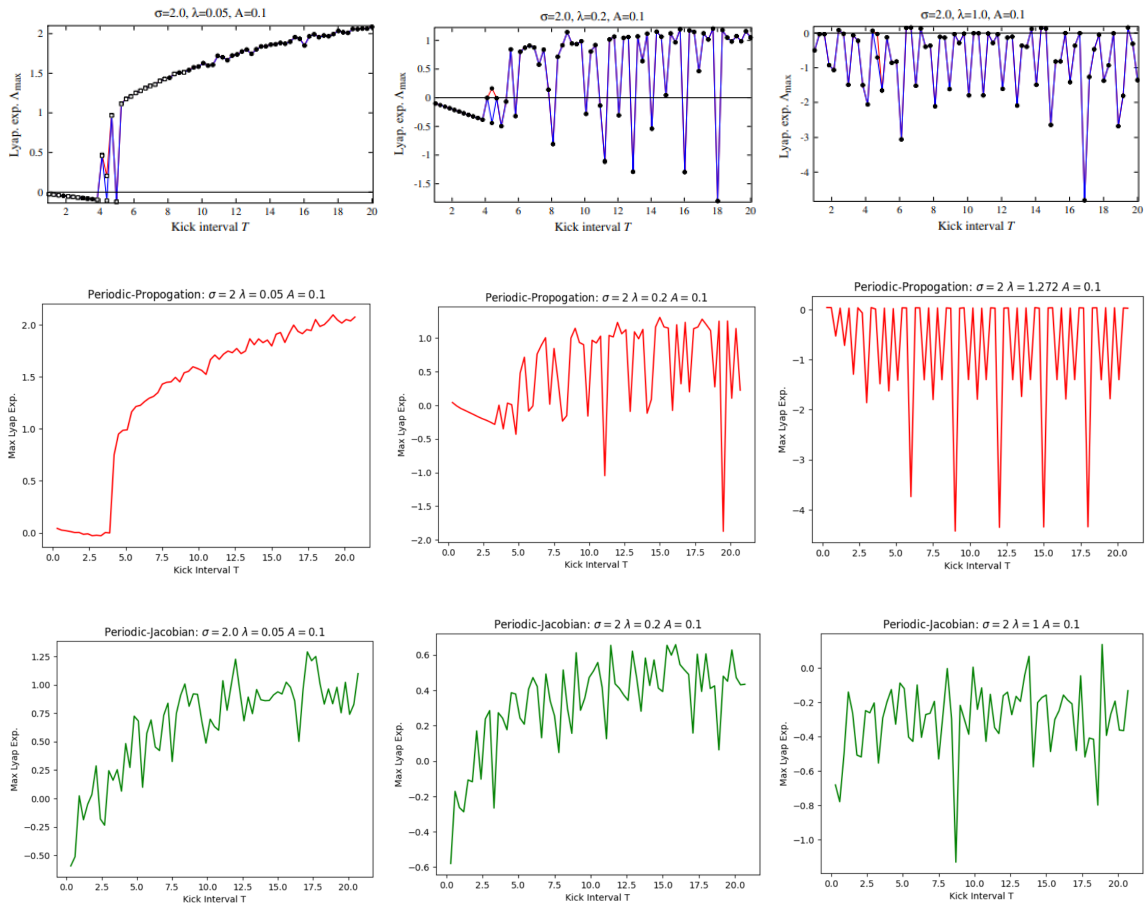

FIGURE 2. Periodic Model: The graphs display the periodic model from the paper in the top row, the propagation replication in the middle row, and the Jacobian replication in the bottom row. Each column exhibits a set of parameters for which we replicated the results for with one slight adjustment to the right most propagation graph.

Unlike with the propagation method, the results from using the Jacobian method align with our expectations. Sustained chaos can be observed with sufficiently large $\frac{\sigma}{\lambda}A$. Here, the mean value of A is used to represent the normally distributed range for our result interpretation. Given that $e^{-\lambda T}$ does not stray too far from 1, we observed that in both sets of parameters (bottom left and middle of Figure 4) chaos is generated, the factor $\frac{\sigma}{\lambda}A(1 - e^{-\lambda T}) \approx 0.6$; and when the sets of parameters have $\frac{\sigma}{\lambda}A(1 - e^{-\lambda T}) < 0.2$, no chaos is speculated. The phenomenon is also in agreement with propositions in [4].

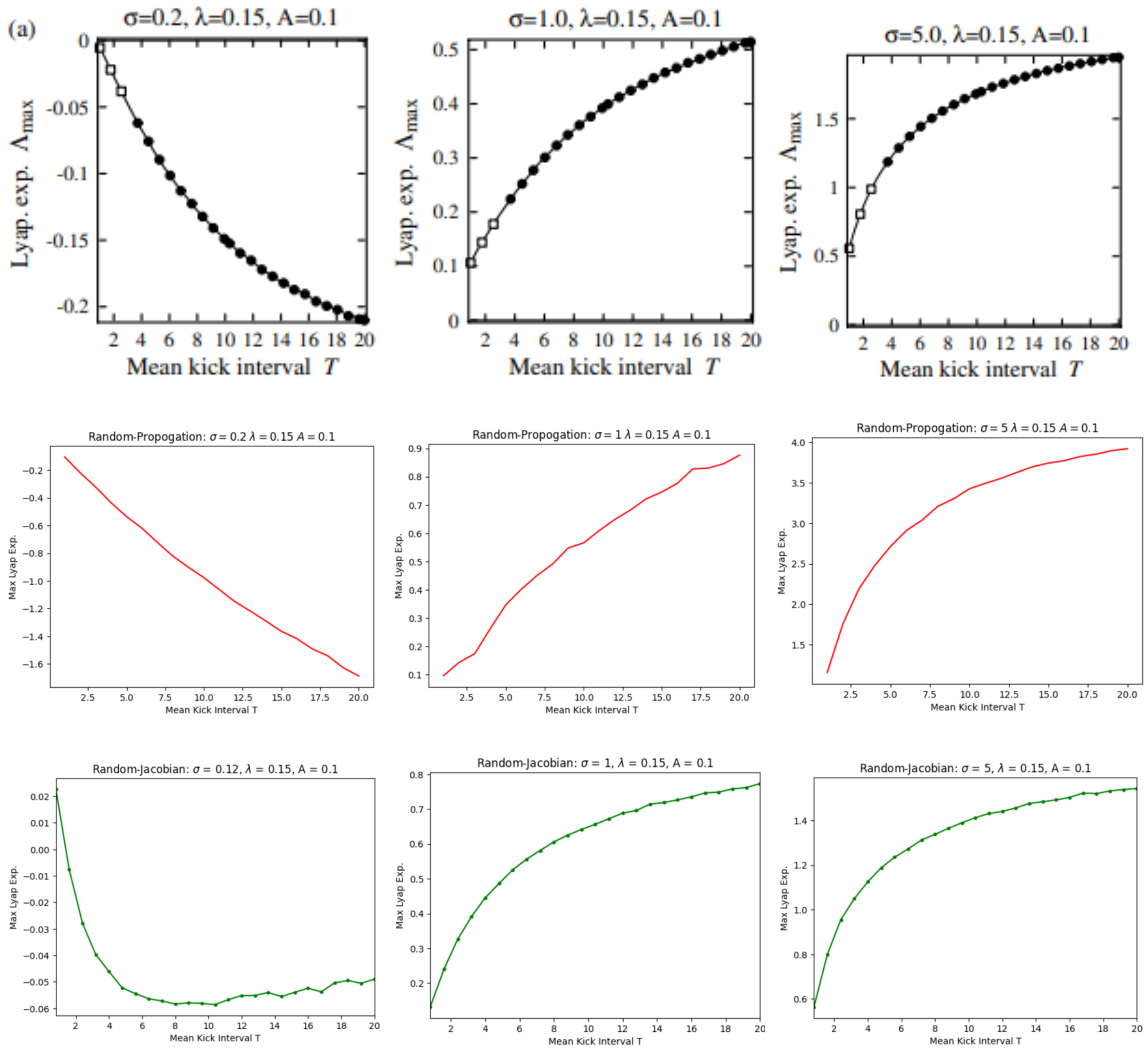

FIGURE 3. Random Model: We have the random model graphs from the paper in the top row, the propagation replication in the middle row, and the Jacobian replication results in the bottom row. Each column has a set of parameters with a slight adjustment to the leftmost graph in the Jacobian method to get near consistent results.

The failure of conformity to previous studies may attribute to sensitive dependence on initial conditions. As stated in Section 2, we chose two arbitrarily close initial tangent vectors, mapped forward and updated new distance over original distance each time step, thus accumulated error in each step accompanied with randomness of amplitudes, which in turn leads to divergent orbital instability. On the other hand, the Jacobian method uses linearization of the current tangent vectors, forward mapping the tangent vectors, and recompute the Jacobian at each time step, thus

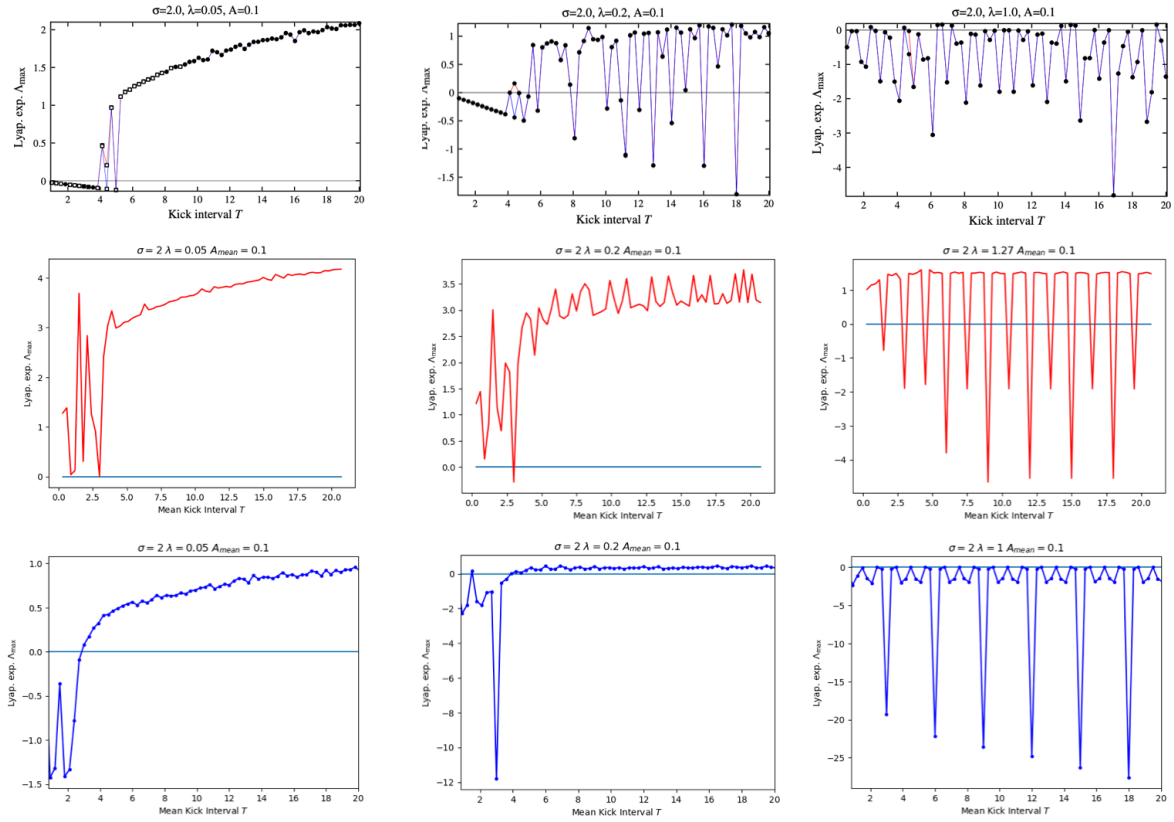

FIGURE 4. The top row shows the periodic model results from study 1 in [3]; the middle row shows the random amplitude with periodic kicks implemented using propagation method as discussed in Section 2, similar general patterns can be observed with all three sets of parameters showing non-conforming chaotic phenomenon; the bottom row shows the random amplitude with periodic kicks implemented using Jacobian method, the results agree with our expectation.

keeping cumulative error relatively low. Therefore, we conclude that the Jacobian method here generates conformed results. As we look further into other explorations, we choose the Jacobian method in implementing our results due to the above-mentioned robustness and accuracy of the method.

3.2. **Widening Distribution of Kicks.** We further investigated the results of a widening distribution of kicks. We generated a distribution of kicks that start tight with growing time between kicks. To keep the average time of the distribution to the specified kick interval we assigned values randomly between 0.1 and the kick interval. We then found the value, which when averaged with

the random value, would return the kick interval. Once we had all the interval values, we sorted them in ascending order. For this analysis, we focused on utilizing the Jacobian method as this was the method used in the paper, and through our investigation was proven to have less variability compared to the propagation method. We ran our model with the new kick intervals and assessed the results in comparison to both the periodic and random models. As we adjusted the parameters, the results mostly match the results of the random model and we are able to produce chaos where the periodic model had no chaos. We were interested in the deviations from the random model and discovered a trend. With the relationship $\frac{\sigma}{\lambda}A \approx 0.16$, the lower average kick intervals greater than 1 produced negative Lyapunov exponents in the widening distribution whereas the random model produced chaos. As the shearing (σ) increased and the relationship $\frac{\sigma}{\lambda}A \approx 0.16$ is maintained, fewer time intervals did not produce chaos.

These results are in Figure 5 which has three sets of graphs, the top row for the periodic model, middle for the random model, and the bottom row for the widening distribution for three sets of parameters which are the columns. It is evident for the periodic model that all sets of parameters produced no chaos, the random model produced only chaos, and the widening distribution is unique. The new distribution has the shape of the random model, but has a period of no chaos in each set of parameters. This is due to the folding relationship, $(\frac{\sigma}{\lambda})A(1 - e^{-\lambda T})$, being small and thus not well formed for $T < 4$. Therefore there is time between kicks for sinks to form thus preventing the strange attractor from developing. For all three widening distributions for approximately $T < 1$ there are positive Lyapunov exponents. We see chaos here because in order to maintain an average time between kicks of this size the kicks remain tight enough to prevent sinks from forming between kicks. These results deviate from the random model where randomness allows tighter bursts of kicks for all T, thus moving the system away from the limit cycle and the strange attractor forms. As we increase the shearing and damping in the system we have fewer time intervals with no chaos. The folding relationship is satisfied for more values of T thus the higher values that initially do not

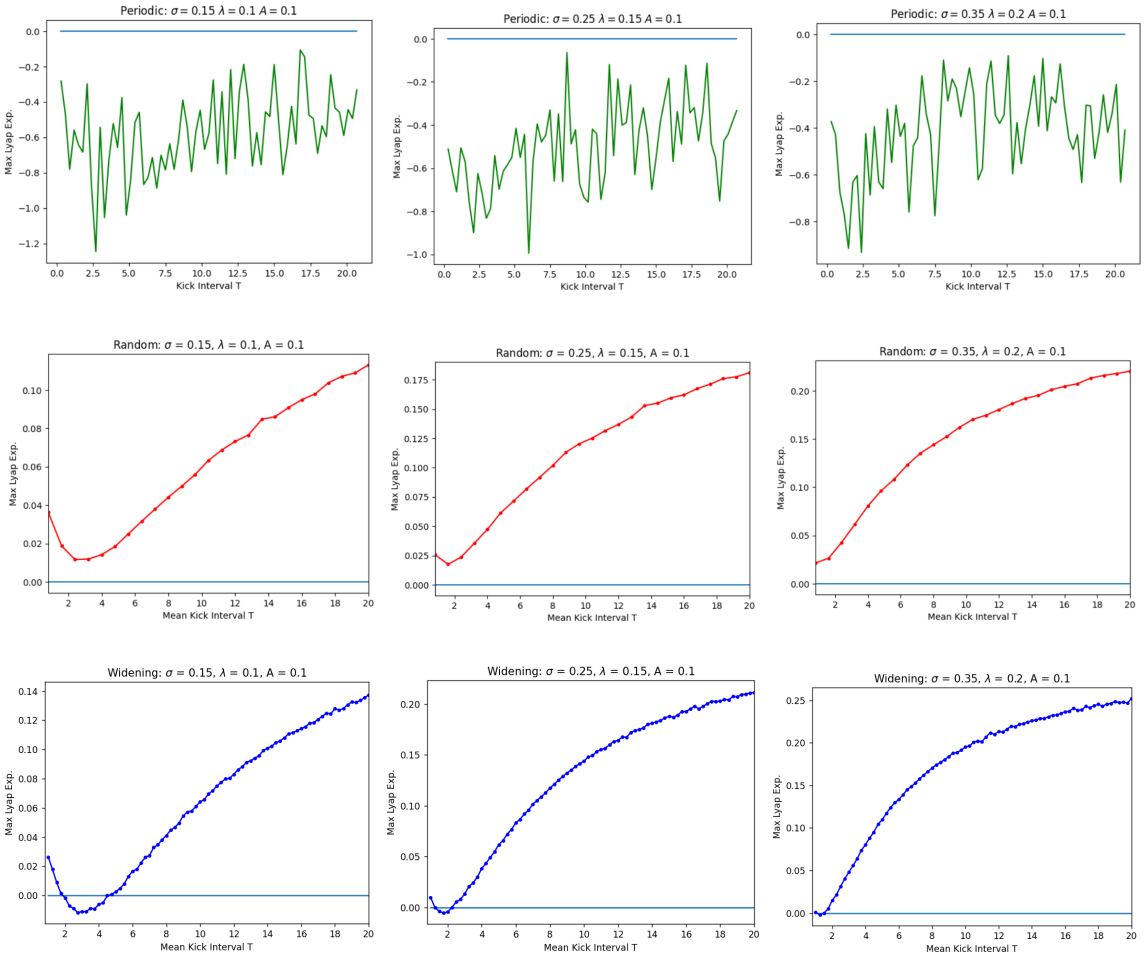

FIGURE 5. We have the graphs for the periodic model in the top row, random model in the middle row, and the widening distribution in the bottom row. Each column displays a set of parameters where we have chaos in all but the periodic model and see variations between the random and widening distributions.

produce chaos start to generate chaos. It would be interesting to continue the exploration to see if there are other relationships between the parameters that produce deviations between the random model and widening distribution model results.

4. Conclusion

In this paper, we have reproduced results for the periodic and random kick studies described in the shear induced chaos paper by Lin and Young. A propagation and Jacobian method were explored for calculating the Lyapunov exponents. Replicated results from both methods were

compared qualitatively to results from the original study, and it was found that the replicated results closely matched a majority of the original results, with a few deviations that may have been due to different numerical methods used. Having successfully reproduced results in the original studies, two new questions were posed and studied.

The first was to see if randomizing the amplitude of kicks in the periodic study would have had an effect on shear-induced chaos. It was seen that the Jacobian method for calculating the Lyapunov exponents mirrored the results from the periodic study, whereas the propagation method had similar results, but with a bias that shifted the values of the Lyapunov exponents above zero. The discrepancy between the replication methods indicated that the Jacobian method was more reliable, thus that method is used for further analysis. The findings were that the introduction of a random amplitude kicks into the periodic kicks did not have a significant impact on inducing chaos.

The second question asked if widening the distribution of random kicks had an additional effect on shear induced chaos. The results from widening the distribution had similar curve of lyapunov exponents when compared to the Poisson random kick study. The only deviation from the Poisson random kick study was that there was a negative bias, bringing some of the smaller T lyapunov exponents below zero. Based on the results, widening the random kicks does have an effect on the shear induced-chaos, and should be a topic that is explored further. An additional study on shear induced chaos could include continuing to modify the distribution of the random kicks either through tighter distributions, or modifying the distribution curvature.

There is a great importance in studies like these to understand when a hyperbolic limit cycle becomes chaotic. There are several real applications, one of which involves controlling chaos in lasers, another is to understand complex biological systems, such as neural networks.

References

[1] Y.-C. Lai and T. Tél. *Transient chaos: complex dynamics on finite time scales*, volume 173. Springer Science & Business Media, 2011.

[2] K. K. Lin, E. Shea-Brown, and L.-S. Young. Reliable and unreliable dynamics in driven coupled oscillators, 2006.

[3] K. K. Lin and L.-S. Young. Shear-induced chaos. *Nonlinearity*, 21(5):899, Mar 2008.

[4] K. K. Lin and L.-S. Young. Dynamics of periodically-kicked oscillators. *Journal of Fixed Point Theory and Applications*, 7(2):291–312, Oct 2010.

[5] E. Shea-Brown and G. Lajoie. Lyapunov exponents. 2023.

[6] J. Sprott. Numerical calculation of largest Lyapunov exponent. 2015.

[7] Q. Wang and L.-S. Young. Strange attractors in periodically-kicked limit cycles and Hopf bifurcations. *Communications in Mathematical Physics*, 240:509–529, 2003.

[8] Wikipedia. Lyapunov exponent — Wikipedia, the free encyclopedia, 2023. [Online; accessed 5-June-2022].

[9] G. Zaslavsky. The simplest case of a strange attractor. *Physics Letters A*, 69(3):145–147, Dec 1978.

Appendix A. Periodic Model using Propagation Method

This list of code contains two variations. The main code is our replication for Periodic Model (study 1) using Propagation Method. The commented code blocks are for our novel results (see Section 3.1 Random Amplitude with Periodic Kicks).

```
import numpy as np
import scipy.integrate
import matplotlib.pyplot as plt

def study1(sigma,lam,A):
############################
## For Novel Result Section 3.1 Random Amplitude with
## Periodic Kicks using Propagation Method
## use the following definition:
## def RandAPeriodK(sig,lam,A_mean)
############################
    t_max=1*10**3
    t=np.linspace(0,t_max,(t_max+1)*3)
    T = np.arange(0.3, 21, .3)

    iter =10
    lyap_temp=np.zeros([iter, len(T)])
    lyap2=np.zeros([len(T)])
    lyap3=np.zeros([len(T)])

    for i in range(len(T)):
        for k in range(iter):

            theta_0 = np.random.uniform(0, .99999999)
            y_0 = np.random.uniform(-0.1, 0.1)

            theta_p0=0.000001
            y_p0=0
            lsum=0
            for j in range(len(t)):
                ###############################################################
                # For Novel Results Section 3.1 Random Amplitude with
                # Periodic Kicks using Propagation Method
                # Add the following Normal Distribution of A
                # A = np.random.uniform (A_mean * 0.9999, A_mean * 1.0001)
                ###############################################################
                #x1
                theta_1 = (theta_0 + T[i] + sigma / lam * ( y_0 + A *
                    np.sin(2 * np.pi * theta_0)) * (1 - np.exp(-lam * T[i]))) % 1
                y_1 = np.exp(-lam * T[i]) * (y_0 + A * np.sin(2 * np.pi * theta_0))

                #x1'
                theta_p1 = (theta_p0 + T[i] + sigma / lam * ( y_p0 + A *
                    np.sin(2 * np.pi * theta_p0)) * (1 - np.exp(-lam * T[i]))) % 1
                y_p1 = np.exp(-lam * T[i]) * (y_p0 + A *
```

```
                    np.sin(2 * np.pi * theta_p0))
                y_diff = (y_p1-y_0)**2
                theta_diff = (theta_p1-theta_0)**2
                df=10**12*(y_diff+theta_diff)
                df = np.sqrt(df)
                rs=1/df
                #forward propagate
                theta_p0=theta_0+rs*(theta_p1-theta_0)
                y_p0=y_0+rs*(y_p1-y_0)

                theta_0=theta_1
                y_0=y_1
                lsum += np.log(df)
            lyap_temp[k, i]=lsum/(len(t))
        ranked=np.sort(lyap_temp[:,i])
        rank_lyap=ranked[1:-2]
        lyap2[i]=np.max(rank_lyap)
        lyap3[i]=np.min(rank_lyap)

    plt.plot(T, lyap2, 'r')
    #plt.plot(T, lyap3, 'r')
    plt.title('Periodic-Propogation: '+ '$\sigma=$'+str(sigma)+
        ' $\lambda=$'+str(lam)+' $A=$'+str(A))
    plt.xlabel('Kick Interval T')
    plt.ylabel('Max Lyap Exp.')
    plt.show()

sigma = 2
lam = .05
A = .1
study1(sigma,lam,A)

sigma = 2
lam = .2
A = .1
study1(sigma,lam,A)

sigma = 2
lam = 1.272
A = .1
study1(sigma,lam,A)
#######################################################
## For Novel Result Section 3.1 Random Amplitude with
## Periodic Kicks using Propagation Method
## Use the Following Parameter and Call function:
## RandAPeriodK(sig = 2, lam = 0.05, A_mean = .1)
## RandAPeriodK(sig = 2, lam = 0.2, A_mean = 0.1)
## RandAPeriodK(sig = 2, lam = 1.27, A_mean = 0.1)
#######################################################
```

Appendix B. Random Model using Propagation Method

This list of code is our replication for Random Model (study 2) using Propagation Method.

```
import numpy as np
import scipy.integrate
import matplotlib.pyplot as plt

sigma = 2
lam = 1
A = .1
t_max = 1 * 10 ** 3
t = np.linspace(0, t_max, t_max + 1)
T = np.arange(1, 21, 1)
lyap = np.zeros([len(T)])
check = np.zeros([len(T)])
tangent = np.zeros([len(T), len(t)])
iter = 10
lyap_temp = np.zeros([iter, len(T)])

y_diff = np.zeros([len(T), len(t)])
theta_diff = np.zeros([len(T), len(t)])
df = np.zeros([len(T), len(t)])
rs = np.zeros([len(T), len(t)])
lyap2 = np.zeros([len(T)])
lyap3 = np.zeros([len(T)])

theta = np.zeros([len(T), len(t)])
theta_prime = np.zeros([len(T), len(t)])
y = np.zeros([len(T), len(t)])
y_prime = np.zeros([len(T), len(t)])

for i in range(len(T)):
    for k in range(iter):
        theta[i, 0] = np.random.uniform(0, .99999999)
        y[i, 0] = np.random.uniform(-0.1, 0.1)
        theta0 = 0.000001
        y0 = 0
        lsum = 0
        for j in range(1, len(t)):
            A_t = np.random.uniform(.8 * A, 1.2 * A)
            T_n=np.random.exponential(scale=T[i])
            theta[i, j] = (theta[i, j - 1] + T[i] + sigma / lam * (
                        y[i, j - 1] + A_t * np.sin(2 * np.pi * theta[i, j - 1]))
                        * (1 - np.exp(-lam * T_n))) % 1
            y[i, j] = np.exp(-lam * T_n) * (y[i, j - 1] + A_t
                * np.sin(2 * np.pi * theta[i, j - 1]))
            theta_prime[i, j] = 1 + sigma * y[i, j]
            y_prime[i, j] = -lam * y[i, j] + A_t * np.sin(2 * np.pi * theta[i, j])
                * dirac(j, i)
```

```
            theta1 = (theta0 + T[i] + sigma / lam * (y0 + A_t
                * np.sin(2 * np.pi * theta0)) * (
                1 - np.exp(-lam * T_n))) % 1
            y1 = np.exp(-lam * T_n) * (y0 + A_t * np.sin(2 * np.pi * theta0))
            y_diff[i, j] = (y1 - y[i, j]) ** 2
            theta_diff[i, j] = (theta1 - theta[i, j]) ** 2
            df[i, j] = 10 ** 12 * (y_diff[i, j] + theta_diff[i, j])
            rs[i, j] = 1 / np.sqrt(df[i, j])
            theta0 = theta[i, j] + rs[i, j] * (theta1 - theta[i, j])
            y0 = y[i, j] + rs[i, j] * (y1 - y[i, j])
            lsum += np.log(df[i, j])
        lyap_temp[k, i] = lsum / len(t)
    ranked = np.sort(lyap_temp[:, i])
    rank_lyap = ranked[1:-2]
    lyap2[i] = np.max(rank_lyap)
    lyap3[i] = np.min(rank_lyap)
plt.title('Random '+ '$\sigma=$'+str(sigma)+
    ' $\lambda=$'+str(lam)+' $A=$'+str(A))
plt.xlabel('Mean Kick Interval T')
plt.ylabel('Max Lyap Exp.')
plt.plot(T, lyap2, 'r')
plt.show()
```

Appendix C. Periodic Model using Jacobian Method

This list of code is our replication for Periodic Model (study 1) using Jacobian Method.

```
import numpy as np
import scipy.integrate
import matplotlib.pyplot as plt

def study1_v2(sigma,lam,A):
    t_max=1*10**3
    t=np.linspace(0,t_max,(t_max+1)*3)
    T = np.arange(0.3, 21, .3)

    iter =10
    lyap_temp=np.zeros([iter, len(T)])
    lyap2=np.zeros([len(T)])
    lyap3=np.zeros([len(T)])

    def jacobian_matrix(theta, T):
        tmp = 2 * np.pi * A * np.cos(2 * np.pi * theta)
        exp_lamT = np.exp(-lam*T)

        j11 = (1 + sigma / lam * tmp * (1 - exp_lamT)) % 1
        j12 = sigma / lam * (1 - exp_lamT)
        j21 = exp_lamT * tmp
        j22 = exp_lamT
        return np.array([[j11, j12],[j21,j22]])
```

```
    for i in range(len(T)):
        for k in range(iter):
            theta_0 = np.random.uniform(0, .99999999)
            y_0 = np.random.uniform(-0.1, 0.1)
            theta_p0=0.000001
            y_p0=0
            lsum=0
            eps0 = np.sqrt((theta_p0 - theta_0)**2 + (y_p0-y_0)**2)

            for j in range(len(t)):
                #x1
                theta_1 = (theta_0 + T[i] + sigma / lam * ( y_0 + A *
                    np.sin(2 * np.pi * theta_0)) * (1 - np.exp(-lam * T[i]))) % 1
                y_1 = np.exp(-lam * T[i]) * (y_0 + A *
                    np.sin(2 * np.pi * theta_0))
                # Form Jacobian matrix
                J1 = jacobian_matrix(theta_1, T[i])
                eigvals =  np.linalg.eig(J1.T*eps0 @ J1*eps0)[0]
                lyap_temp1 = np.log(np.sqrt(np.abs(np.max(eigvals))))
                lsum += lyap_temp1
                theta_0=theta_1
                y_0=y_1

            lyap_temp[k, i]= np.max(lsum/(len(t)*2))
        lyap2[i]=np.max(lyap_temp[:,i])
        lyap3[i]=np.min(lyap_temp[:,i])

    plt.plot(T,np.zeros(np.shape(lyap2)))
    plt.xlabel('Kick Interval T')
    plt.ylabel('Max Lyap Exp.')
    plt.plot(T,lyap2,'g')
    plt.title('Periodic: $\sigma=$'+str(sigma)+' $\lambda=$'+str(lam)+'
        $A=$'+str(A))
    plt.show()

sigma = 0.35
lam = .2
A = .1
study1_v2(sigma,lam,A)

sigma = .25
lam = .15
A = .1
study1_v2(sigma,lam,A)

sigma = .15
lam = .1        # in paper lambda=1
A = .1
study1_v2(sigma,lam,A)
```

Appendix D. Random Model using Jacobian Method

This list of code contains two variations. The main code is our replication for Random Model (study 2) using Jacobian Method. The commented code blocks are for our novel results (see Section 3.1 Random Amplitude with Periodic Kicks).

```
import numpy as np
import matplotlib.pyplot as plt

def study2(sig, lam, A_mean):
############################
## For Novel Result Section 3.1 Random Amplitude with
## Periodic Kicks using Jacobian Method
## use the following definition:
## def RandAPeriodK_v2(sigma,lam,A_mean)
############################

    T_means = np.arange(0.8, 20.7, 0.8)
    n_iterations = 4 * 10 ** 3
    eps = 1e-8
    iter = 10

    lyap_temp = np.zeros([iter,len(T_means)])
    lyap_max = np.zeros(len(T_means))
    lyap_min = np.zeros(len(T_means))

    # define jacobian matrix using eq (5)
    def jacobian_matrix(theta, T, A):
        tmp = 2 * np.pi * A * np.cos(2 * np.pi * theta)
        exp_lamT = np.exp(-lam * T)

        j11 = (1 + sig / lam * tmp * (1 - exp_lamT)) % 1
        j12 = sig / lam * (1 - exp_lamT)
        j21 = exp_lamT * tmp
        j22 = exp_lamT

        return np.array([[j11, j12],[j21,j22]])

    for i in range(len(T_means)):
        for k in range(iter):
            T_mean = T_means[i]

            # initial value of theta and y
            thetas = np.zeros([n_iterations])
            ys = np.zeros([n_iterations])
            Ts = np.zeros([n_iterations])
            thetas[0] = np.random.uniform(0, 1-eps)
            ys[0] = np.random.uniform(-0.1, 0.1)
            lsum = 0

            for j in range(1, n_iterations):
```

```
                T = np.random.exponential(T_mean)
                ##########################################################
                ## For Novel Result Section 3.1 Random Amplitude with Periodic Kicks
                ## Used the following instead:
                ## T = T_mean
                ##########################################################
                A = np.random.uniform(0.8 * A_mean, 1.2 * A_mean)
                tmp = ys[j-1] + A * np.sin(2 * np.pi * thetas[j-1])
                exp_lamT = np.exp(-lam * T)
                thetas[j] = ( thetas[j-1] + T + sig / lam * tmp
                    * (1 - exp_lamT) ) % 1
                ys[j] = tmp * exp_lamT
                Ts[j] = Ts[j-1] + T
                eps0 = np.sqrt((thetas[j] - thetas[j-1])**2 + (ys[j]-ys[j-1])**2)

                J = jacobian_matrix(thetas[j], T, A)

                # Eigenvalue of Jacobian
                eigvals = np.linalg.eig((J.T*eps0) @ (J*eps0))[0]
                lsum += np.log(np.abs(np.max(eigvals)))/2

            lyap_temp[k, i]= lsum / (len(t)*2)

        lyap_max[i]=np.max(lyap_temp[:,i])
        lyap_min[i]=np.min(lyap_temp[:,i])

    plt.title(f'$\sigma$ = {sig}, $\lambda$ = {lam}, A = {A_mean}')
    plt.plot(T_means,np.zeros(np.shape(lyap_max)))
    plt.plot(T_means, lyap_max,'b.-')
    plt.xlim([0.8, 20])
    plt.xlabel('Mean Kick Interval $T$')
    plt.ylabel('Lyap. exp. $\Lambda_\max{}$')
    _, _ = plt.xticks(np.arange(2, 21, 2))
    plt.show()

study2(sig = .35, lam = .2, A_mean = .1)
study2(sig = .25, lam = .15, A_mean = .1)
study2(sig = .15, lam = .1, A_mean = .1)
#####################################################
## For Novel Result Section 3.1 Random Amplitude with
## Periodic Kicks using Jacobian Method
## Use the Following Parameter and Call function:
## RandAPeriodK_v2(sigma = 2, lam = 0.05, A_mean = .1)
## RandAPeriodK_v2(sigma = 2, lam = 0.2, A_mean = .1)
## RandAPeriodK_v2(sigma = 2, lam = 1, A_mean = .1)
#####################################################
```

Appendix E. Widening Distribution Using Jacobian Method

This list of code is our Novel Result Section 3.2 Exploration of Widening Distribution Using Jacobian method.

```
import numpy as np
import matplotlib.pyplot as plt
import random
# Parameters
sig = .35
lam = .1
A_mean = .1
T_means = np.arange(1, 21, .25)
n_iterations = 4 * 10 ** 3
eps = 1e-8

def jacobian_matrix(theta, T, A):
    tmp = 2 * np.pi * A * np.cos(2 * np.pi * theta)
    exp_lamT = np.exp(-lam * T)

    j11 = (1 + sig / lam * tmp * (1 - exp_lamT)) % 1
    j12 = sig / lam * (1 - exp_lamT)
    j21 = exp_lamT * tmp
    j22 = exp_lamT
    return np.array([[j11, j12], [j21, j22]])

lyap_max = np.zeros(len(T_means))
lyap_sum1 = np.zeros(len(T_means))

for i in range(len(T_means)):

    T_mean = T_means[i]
    thetas = np.zeros([n_iterations])
    ys = np.zeros([n_iterations])
    Ts = np.zeros([n_iterations])
    lyaps1 = np.zeros([n_iterations])
    thetas[0] = np.random.uniform(0, 1 - eps)
    ys[0] = np.random.uniform(-0.1, 0.1)
    T_dist = np.zeros([n_iterations])
    half = int(len(T_dist) / 2)
    for z in range(half):
        if len(T_dist) % 2 != 0:
            z += 1
        T_dist[z] = np.random.uniform(.1, T_means[i])
        T_dist[half + z] = (T_means[i] * 2) - T_dist[z]
    if len(T_dist) % 2 != 0:
        T_dist[half] = T_means[i]
    T_dist = np.sort(T_dist)
#############Alternate T_dist widening, same results#########
  #  T_dist = np.zeros([n_iterations])
  #   half = int(len(T_dist) / 2)
  #   T_dist[0:half]=np.arange(0.1, T_mean+0.1, T_mean/half)
  #   if len(T_dist) % 2 != 0:
  #       T_dist[half] = T_mean
```

```
  #   for z in range(half):
  #       # if len(T_dist)%2 !=0:
  #       #     z+=1
  #       T_dist[len(T_dist)-z-1] = (T_mean * 2) - T_dist[z]
###############Constant A same results#########################

    for j in range(1, n_iterations):
        T = T_dist[j]
        A = np.random.uniform(0.8 * A_mean, 1.2 * A_mean)
        tmp = ys[j - 1] + A * np.sin(2 * np.pi * thetas[j - 1])
        exp_lamT = np.exp(-lam * T)
        thetas[j] = (thetas[j - 1] + T + sig / lam * tmp * (1 - exp_lamT)) % 1
        ys[j] = tmp * exp_lamT
        Ts[j] = Ts[j - 1] + T

        # Form Jacobian matrix
        J = jacobian_matrix(thetas[j], T, A)

        # Singular value of Jacobian
        eigvals = np.linalg.eig(J.T @ J)[0]
        lyap_temp1 = np.log(np.sqrt(np.abs(np.max(eigvals))))
        lyaps1[j] = lyap_temp1

    lyap_sum1[i] = np.cumsum(lyaps1)[-1] / (2*n_iterations)

plt.plot(T_means,np.zeros(np.shape(lyap_max)))
plt.title(f'Widening: $\sigma$ = {sig}, $\lambda$ = {lam}, A = {A_mean}')
plt.xlabel('Mean Kick Interval T')
plt.ylabel('Max Lyap Exp.')
plt.plot(T_means, lyap_sum1, 'b.-')
plt.xlim([0.8, 20])
_, _ = plt.xticks(np.arange(2, 21, 2))
plt.show()
```