A Topological Assessment of the RK4 Method for Chaotic Driven Damped Pendulums

Oliver Burch

School of Mathematical and Physical Sciences, University of Sheffield

March 30, 2026

Abstract

In this article we assess the performance of the RK4 method at modelling the chaotic dynamics of a simulated damped driven pendulum. Chaotic systems, while very well studied, continue to pose significant predictive challenges for mathematicians. It is therefore imperative that we have a strong model to make these predictions at a high level of accuracy. We build our model by benchmarking our solver with the known mass-on-a-spring analytical solutions and then run three simulations of varying resolution for the RK4 to compare results of topological features of the dynamics. We show that the RK4 successfully captures the topological features of the chaos - including braiding, unstable periodic orbits, and a conserved correlation dimension of $\simeq 0.8$ - and validate its use through the result of positive Lyapunov exponents across all resolutions. The end the article gives a comparative assessment of increasing the resolution against the computational strain of doing this and compares our work with other numerical solvers.

Contents

1 Introduction

Dynamical systems have been a topic of discussion amongst intellectuals for many millennia. The ancient geometrical approach of the Greeks to these systems varies greatly from our modern approach of calculus but nonetheless these still prove to be a foundational part of our understanding of the world. Modern dynamics emerges from Henri Poincaré's work on the three body problem [1] in the 1890s and was formalised with Edward Lorenz's work on chaotic dynamics in the 1960s [2].

Many different types of dynamics emerge from the studies of a variety of systems. We may find differential equations in the formulas for fluid dynamics or Hamiltonian dynamics in quantum theory. For the purpose of this paper we will be discussing only the systems involving differential equations.

There are many classes of differential equations. One of these is the linearity of an equation. We define a differential equation as being linear if the power of a variable only appears once with no function acting on the variable being classified as part of the equation, else the equation becomes non-linear. Some examples of linear equations include: mass on a spring systems, population growth - or as we will see in the theory section of this paper - the undamped pendulum. In contrast, some non-linear equations are the Navier-Stokes equations and the driven damped pendulum.

Linear equations typically have exact solutions. We have numerous mathematical tools developed to solve these such as power series solutions. Contrary to this, non-linear dynamics is a difficult area of mathematics to work in. They often have no analytical solution and require numerical solvers to make approximations of the numerical values of the system. For example, the Navier-Stokes equations are part of a set of problems called the Millennium Prize Problems. Finding solutions to these equations would grant the mathematician $1,000,000 and the privilege of having their name known as one of the greats.

We can use numerical solvers to find approximations for these systems with non-analytical solutions. A wide variety of these exist and are typically specialist in their usage, namely they are quite circumstantial. A Newton-Raphson solver may be used to approximate solutions (roots) to complex equations while a Runge-Kutta solver may be used to solve an ordinary differential equation. A Bulirsch-Stoer solver might then be used to solve perfectly smooth systems to greater orders of accuracy. The ideas behind these solvers will be fundamental to this paper, especially the case of the Runge-Kutta as it is the sole subject of this writing.

The rest of this introduction will focus on the core motif of this paper: to what extent is the RK4 method a valid numerical approach for the topological assessment of the chaotic

driven damped pendulum? To be able to assess this we first need an understanding of the pendulum and where it comes from.

In 1656, Christiaan Huygens attempted to reduce the error of the 'modern' clock. Before his time clocks would lose 15 minutes per day [6]. As a physicist and astronomer, Huygens realised this was causing large discrepancy in his work and set out to minimise this by introducing dependence on the motion of a pendulum swing to move the clock mechanism. This work was strongly inspired by the work of Galileo Galilei in the early 1600s and has since been widely considered as the most important use of the pendulum. Huygens' clock reduced the error of 15 minutes to 15 seconds and set a new standard of measurements.

Today, the pendulum is a widely studied system which forms the foundations of many physics textbooks. One such textbook is J. R. Taylor's *Classical Mechanics* [7] wherein the driven damped pendulum is used as the core study of chaotic systems. In 1963, Edward Lorenz published what is widely accepted as the first analysis of chaotic systems, 'Deterministic Nonperiodic Flow' [2]. In this text, Lorenz made the claim that nonperiodic solutions cannot readily be determined by numerical procedures, and this forms the basis of our paper.

We can imagine a non-damped pendulum as one such that throughout all time and space, so long as no external force is applied to the system, the period of the system will remain constant. This is clearly a linear system. So long as we know where the pendulum is, we can predict the next point it will be at. However, what if we add a damping term like air resistance, and a driving term such as a driven pivot? Could we still predict the next step of the system from any point we choose to start?

Chaos theory emerges in a large number of our day-to-day experiences. We use the Navier-Stokes equations to predict the weather, making long range forecasts particularly difficult to estimate. We use stress conditions to exhibit chaotic dynamics in bacterial growth and gene expression. We can use specific nonlinear boundary conditions for the wave equation to simulate chaotic traffic jams.

With chaotic dynamics dictating such a large number of our natural (and unnatural) systems, it is incredibly important to have strong analytical methods to capture the useful information from these. The definition of chaos is: the property of a complex system whose behaviour is so unpredictable as to appear random, owing to great sensitivity to small changes in conditions. Following this definition it is essential that our metric of measuring chaos has minimal divergence from the actual value in order to best numerically approximate the system.

This paper will attempt to judge how justifiable the RK4 solver is by following this practise. In the modern age, chaos is still being studied in robotics, cryptography, biology,

politics and elsewhere. It is therefore necessary to have a robust model to build chaotic predictions with high levels of accuracy, while maintaining a feasible computation time. We will attempt to construct the theory and computational methods needed to achieve this, then will present our results with a discussion comparing a fidelity analysis - involving Lyapunov exponents and the correlation dimension - and a computational analysis of multiple resolutions of the RK4 solver. We will end the paper with a final verdict and some improvements for future work in this field.

2 Theory

2.1 The Runge-Kutta Fourth-Order Method

"The idea of Euler was to propagate the solution of an initial value problem forward by a sequence of small time-steps. In each step, the rate of change of the solution is treated as constant and is found from the formula for the derivative evaluated at the beginning of the step." This is an excerpt from 'A History of Runge-Kutta Methods' [3]. It directly captures the brilliance of Leonhard Euler and his numerical solver for non-linear ordinary differential equations. In the late 1700s, Euler had been troubled by issues with equations he was facing in ballistics having no analytical solution. He believed mathematics could predict the solutions to these and as such developed the solution:

$$y_n = y_{n-1} + (x_n - x_{n-1})\, f(x_{n-1},\, y_{n-1}) \tag{1}$$

Where $y'(x) = f(x, y(x))$. In the late 1800s, Carl Runge looked to improve this method by averaging the slope of multiple points taken between Euler's two points x_n and x_{n-1}. For an s-stage Runge-Kutta model:

$$y_{n+1} = y_n + hb_i k_i \tag{2}$$

Where k_i is given by:

$$k_i = f(t_n + c_i h,\ y_n + ha_{ij}k_j) \tag{3}$$

Here: s is the number of stages, h is the step size $x_n - x_{n-1}$, k_i is the slope at a stage i, b_i is the weights of the slope, t_n is the independent variable of the system (here representing time), c_i is the time step fractions, a_{ij} are the coefficients determining how the previous slope affects the current. For use in this project we consider s = 4 for the RK4 model such that:

$$y_{n+1} = y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{4}$$

2.2 Evolution of the Pendulum Model

We now turn our attention to the pendulum. We first consider the case of no damping such that our system is explicitly determined by the initial angle θ_0 and initial velocity ω_0. The system is modelled by the gravitational force and as such we have:

$$\frac{d^2\theta}{dt^2} = -\frac{g}{l}\sin(\theta) \tag{5}$$

Which when using a small angle approximation can be considered to have oscillatory solutions:

$$\theta(t) = \theta_0 \cos(t) \tag{6}$$

Where g is the gravitational constant, and l is the length of the pendulum. Then we add a damping term 'β' such that:

$$\frac{d^2\theta}{dt^2} = -\frac{g}{l}\sin(\theta) - \beta\frac{d\theta}{dt} \tag{7}$$

Where β might be physically interpreted as any constant working against the system. An example of this could be air resistance as the pendulum swings. There is no universal analytical solution to this equation but we can consider the cases of underdamping, critically damping or overdamping where we can calculate the value of the damping coefficient β needed for critical damping by again acknowledging the small angle approximation on a syste, for g = l = 9.81:

$$\beta_c^2 - 4\left(\frac{g}{l}\right) = 0 \tag{8}$$

$$\beta_c = +2 \tag{9}$$

Where we only consider positive coefficients to allow for β to be a damping coefficient and not a driving coefficient. To add a driving term to the system, the equation instead takes the form:

$$\frac{d^2\theta}{dt^2} = -\frac{g}{l}\sin(\theta) - \beta\frac{d\theta}{dt} + A\cos(\omega_D t) \tag{10}$$

Where A is the driving amplitude and ω_D is the driving frequency. This is an equation that has been deeply studied and has no analytical solutions. It is the effort of the rest of this article to convince the reader that we can model the numerical solutions to this equation with some high level of accuracy using the RK4 numerical solver.

2.3 Poincaré Sections and Bifurcation

A first step in understanding this equation is by studying the Poincaré section of the system at a fixed time. We briefly sidetrack from this to describe phase space - an abstract space where each axis represents a variable needed to completely define the

systems current space. In our case this is θ and ω. We now have to first extend our non-autonomous system, with a dependence on time, to an autonomous system by treating the driving phase as a third spatial dimension $\phi = \omega_D t$ such that:

$$\underline{X} = (\theta, \omega, \phi) = \Phi_t(\underline{X}_0) \tag{11}$$

Where Φ_t is the flow function. We infer from the name that the Poincaré section is an (n-1) dimensional hypersurface 'Σ' within an n dimensional phase space. The Poincaré map $P : \Sigma \to \Sigma$ is then the operator that maps the k-th intersection to the (k+1)-th intersection:

$$\underline{X}_{k+1} = P(\underline{X}_k) = \Phi_{\tau(\underline{X}_k)}(\underline{X}_k) \tag{12}$$

Where $\tau(\underline{X}_k)$ is the first return time, namely the time taken for the trajectory of the system of starting point $\underline{X}_k$ to return to the hypersurface. Here, as our system is stroboscopic, $\tau(\underline{X}_k) = \frac{2\pi}{\omega_D}$.

Now instead imagine this concept with a control parameter 'A', we can redefine the mapping operator as:

$$\underline{X}_{n+1} = P(\underline{X}_n; A) \tag{13}$$

If we start at a low driving amplitude (that is $A \simeq 0$) then we have a stable Period-1 fixed point $\underline{X}^* = P(\underline{X}^*; A)$ where the stability of this system is determined by the eigenvalues of the Jacobian matrix evaluated at $\underline{X}^*$:

$$J(\underline{X}^*; A) = \left.\frac{\partial P}{\partial X}\right|_{X=X^*} \tag{14}$$

Here a bifurcation point A_c is one such that $|\Lambda(A_c)| = 1$ where by the stability of $\underline{X}^*$ is lost and an unstable periodic orbit is created. For the driven damped pendulum $\Lambda(A_c) = -1$. This is known as period doubling and our results will hopefully mimic this behavior. This flip causes a stable and an unstable trajectory known as the Period-2 orbit. The same behavior is typical of these new trajectories and as such we can get Period-n orbits where n belongs to the set of 2^m integers. The parameter intervals between these bifurcations shrinks according to the Feigenbaum constant and so these orbits converge to the limit known as the accumulation point. At this point (and for a small segment beyond), stable orbits are destroyed, the Smale horseshoe is formed and the largest Lyapunov exponent turns positive. As such regions of deterministic chaos are created known as strange attractors which we will come to measure using the correlation dimension. These points will be discussed in more detail in the subsequent sub-sections of this theory analysis.

2.4 Dimensions of Chaos - Fidelity

Rényi Dimensions

The Rényi Entropy can be understood as follows. Imagine covering our newly collated Poincaré section with a grid of boxes, all of side length r. If p_i is the probability of finding a point in the ith box - given by $\frac{N_i}{N}$ with N_i = Number of points in ith box - then the Rényi entropy of order q is given by:

$$H_q = \frac{1}{1-q} \ln \sum_i p_i^q \tag{15}$$

We can use this result to derive the spectrum of generalized dimensions by taking $r \to 0$ and logically counting the boxes:

$$D_q = \lim_{r \to 0} \frac{H_q}{-\ln(r)} = \lim_{r \to 0} \frac{1}{q-1} \frac{\ln \sum_i p_i^q}{\ln(r)} \tag{16}$$

Where q will take the physical form of 0, 1 or 2 (but is not limited to any of these and can in fact be any real number). We will explicitly consider the result for q = 2.

Correlation Dimension

The case of q = 2 is given the name the correlation dimension. It is a special case of the Rényi dimensions and is determined by the result:

$$D_2 = \lim_{r \to 0} \frac{\ln \sum_i p_i^2}{\ln(r)} \tag{17}$$

Where p_i^2 is the probability of 2 particles falling into the same box. This is particularly useful to help understand the correlation dimension. We now instead turn to the Grassberger-Procaccia algorithm [4]. Imagine we consider the strange attractor of the system with N points. We can choose a point x_i to start at and draw a ball of radius 'r' around it. Finally we count how many other points x_j fall inside the ball. Doing this for every point on the attractor results in the correlation integral - the probability that any two randomly chosen points are separated by a distance less than r:

$$C(r) = \lim_{N \to \infty} \frac{2}{N(N-1)} \sum_{i<j} \Theta(r - |x_i - x_j|) \tag{18}$$

The function $\Theta(r - |x_i - x_j|)$ is given the name the Heaviside step function and outputs 1 if $|x_i - x_j| < 0$, or 0 otherwise. This means that terms outside the ball are ignored for the specific point x_i as we required.

Now making r very small and following some logical interpretation of the system:

$$C(r) \propto r^D \tag{19}$$

Such that:

$$D = \lim_{r \to 0} \frac{\ln C(r)}{\ln(r)} \tag{20}$$

And we can make the connection that the specific case of q = 2 in the Rényi entropy is actually the same as the correlation dimension produced by the Grassberger-Procaccia algorithm ($D = D_2$ and $C(r) = \sum_i p_i^2$ in the limit of $r \to 0$).

Lyapunov Exponents

In chaotic systems, there exists a set of exponents equal in size to the number of dimensions in the system. These exponents are the Lyapunov exponents and are calculated using the Kaplan-Yorke conjecture:

$$D_{KY} = j + \frac{\lambda_i}{|\lambda_{j+1}|} \tag{21}$$

Where $D_{KY} \approx D_1$ in our Rényi entropy. For the sake of simplicity, and as it does not change the fidelity of the system, this paper will only consider the maximum exponent λ_1, that is the axis that stretches the fastest:

$$\lambda_1 = \lim_{t \to 0} \frac{1}{t} \ln \left(\frac{|\delta x(t)|}{|\delta x_0|} \right) \tag{22}$$

This exponent explicitly tells us how fast the system is exhibiting chaos which is a result we will need in the discussion section.

2.5 Topology

Topological Folding

To assess the validity of using the RK4 to correctly define chaos, we will also need to model the topology of the system. We can first consider the folding of the system. This consists of two elements: the dissipation and the stretching. We define each separately where from the pendulum equations we can calculate the dissipation using the Liouville formula:

$$\frac{dV}{dt} = (\nabla \cdot \underline{F})V = -\beta V \tag{23}$$

Where F is the vector field $F = (\dot{\theta}, \dot{\omega}, \dot{\phi})$ and V is the volume of the phase space. Solving this leads to the result:

$$V(t) = V_0 e^{-\beta t} \tag{24}$$

Which predicts that as $t \to \infty$ the volume of the phase space collapses onto the strange attractor. Now taking our Poincaré section 'P', and letting $X_n = (\theta_n, \omega_n)$ we find:

$$X_{n+1} = P(X_n) \tag{25}$$

Where 'P' acts to map one point to the next. Conversely:

$$J(\underline{X}^*) = \begin{pmatrix} \frac{\partial \theta_{n+1}}{\partial \theta_n} & \frac{\partial \theta_{n+1}}{\partial \omega_n} \\ \frac{\partial \omega_{n+1}}{\partial \theta_n} & \frac{\partial \omega_{n+1}}{\partial \omega_n} \end{pmatrix} \tag{26}$$

Where X^* is a fixed point of the section 'P'. The eigenvalues of this equation will be equivalent to the stretch of the system with $\Lambda_1 > 1$ being the numerical factor that the system grows between driving periods. To avoid the system having infinite energy as $n \to \infty$, P must act as a diffeomorphism, periodically mapping a domain D back onto itself. Namely

$P(D) \cap D$ results in folded, disconnected strips.

Unstable Periodic Orbits

Finally, if we take the collection of these strips, we find that for the equation:

$$\theta_{n+1} = \theta_n \tag{27}$$

There is a distinct set of solutions known as the Unstable Periodic Orbits (UPOs):

$$f(\theta_n) - \theta_n = 0 \tag{28}$$

Where $f(\theta_n)$ can be approximated using the Newton-Raphson iteration formula. For this system these are the Period-1 UPOs, a selection of solutions where tiny changes in the measurements of the system would result in chaotic shifts from their orbit. When there are an infinite number of these UPOs, we form a Smale horseshoe as was discussed in the bifurcations subsection.

3 Methods

3.1 RK4 Implementation

After such a mathematically heavy theory section, it is finally time to discuss the computational methods behind this project. The entire code for this project can be found in the appendix of this article. We start this as we started the theory section, with the

discussion of the RK4. Using the basic mathematical principle set out in Equation 4, we can build a solver by establishing our x and y values, and our function and step size. A large part of the end discussion of this paper will be focused on how the step size affects our results of the fidelity analysis and so it is important we get this correct here.

3.2 Benchmarking

To do this we can benchmark our algorithm against known analytical solutions to differential equations. As RK4 is a numerical solver we can apply it to any system of our choosing so here use the example of a mass on a spring. The analytical solutions follow:

$$x(t) = x_0 \cos(t), \qquad v(t) = -x_0 \sin(t) \tag{29}$$

And we parse the differential equations:

$$\frac{dx}{dt} = v, \qquad \frac{dv}{dt} = -\frac{k}{m}x \tag{30}$$

Through the RK4 algorithm. Here, k is the spring constant and m is the mass. Both are set to 1 for the sake of simplicity in this solver. The result as seen in Fig 1 follows that visually our solver perfectly calculates the trajectories of the analytical solutions, however this is not entirely the case as Fig 2 presents the error in position as time continues. This was calculated by taking the difference between each point of the benchmark and RK4 plots. This error emerges from the truncation of attempting to reduce an infinite series to one that is finite. As the truncation of RK4 scales as $O(h^4)$ (where h is step size) our comparison for different resolutions of plots using the RK4 (see the Poincaré subsection) will include vastly different errors. Fig 3 shows the decrease in error as we scale our step smaller by a factor of 10, namely the truncation shrinks by a factor of $x10^4$ as we would expect. This is, however, not limiting for this paper as we are assessing the topological and statistical features of the strange attractor and therefore while co-ordinates may shift, the measures we take are well defined globally.

3.3 Phase-Space Plots

Using the results from Equation 10, we can compute the pendulum by parsing the function:

$$\frac{d\theta}{dt} = \omega, \qquad \frac{d\omega}{dt} = -\frac{g}{l}\sin(\theta) - \beta\omega + A\cos(\omega_D t) \tag{31}$$

Through the RK4 solver we built in the previous section. We can check this computes the expected result by checking the damping of the system matches what we'd expect it to. From Equation 9, and letting g = 9.81, l = 9.81 and $\omega_D = 0.67$ and A = 0 for simplicity, we can calculate $\beta_c = 2$. We define the vector field for this system as being equal to the

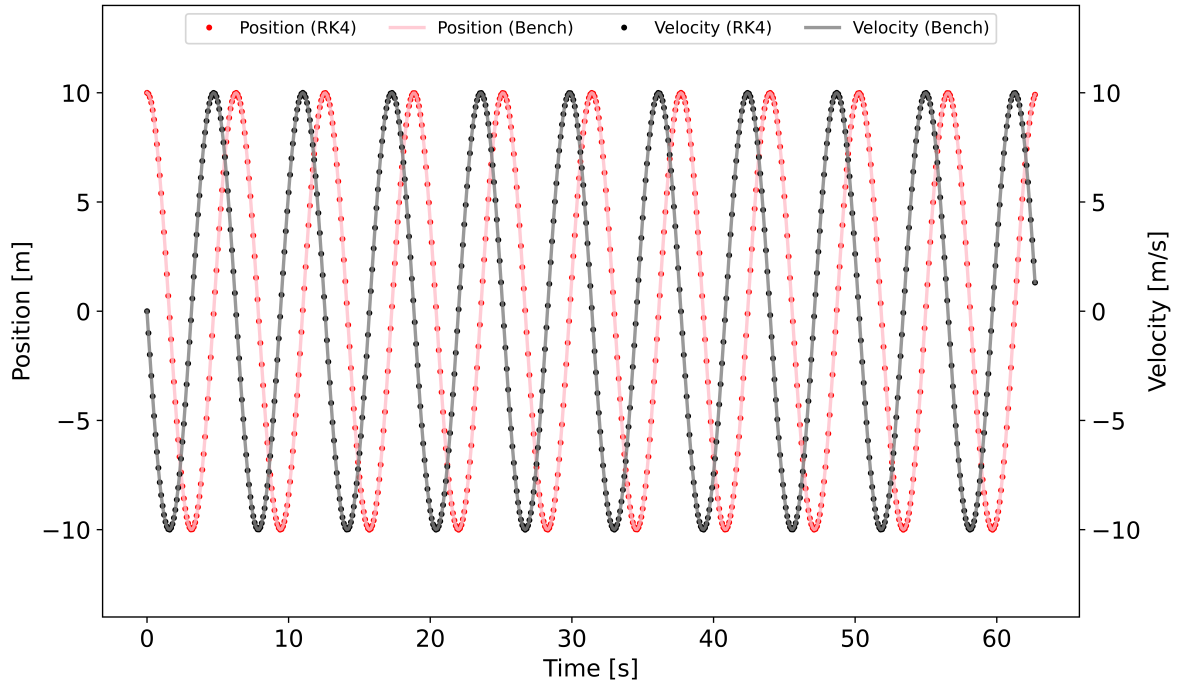

Figure 1: Assessing the validity of our RK4 numerical solver by plotting it for steps of h = 0.1 against the analytical solution to the mass on a spring system. Here is is clear that our solver provides a very good estimate of how the system behaves for the initial condition $x_0 = 10$ as there is no visible divergence from the curve.

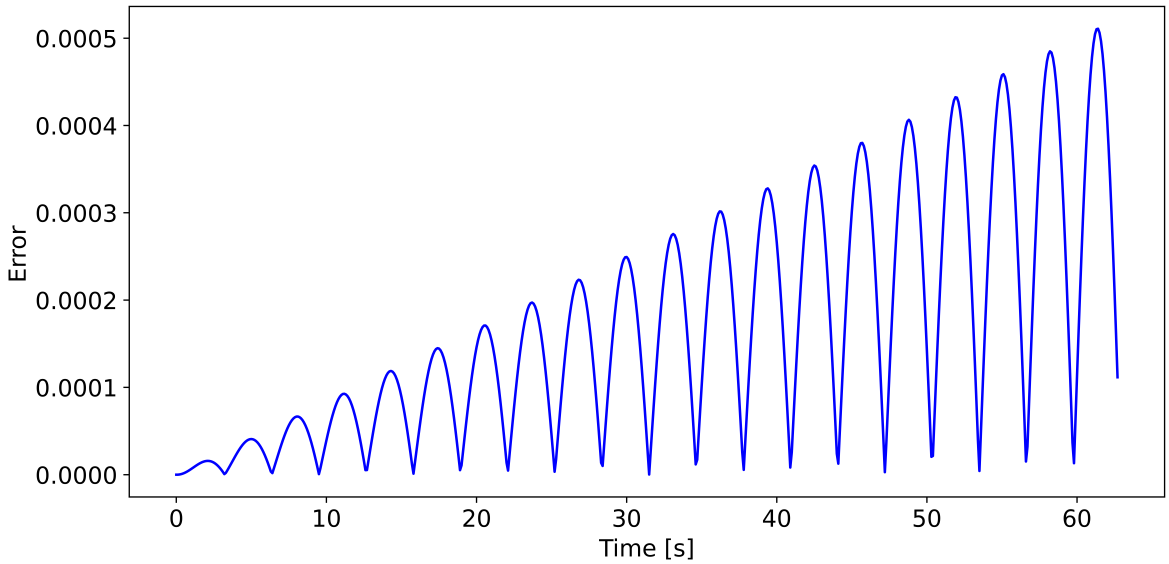

Figure 2: The error for the RK4 method with h = 0.1 is shown for the mass-spring system. The error is produced from the truncation as $O(h^4)$. It was calculated by plotting the difference in the analytical and numerical results.

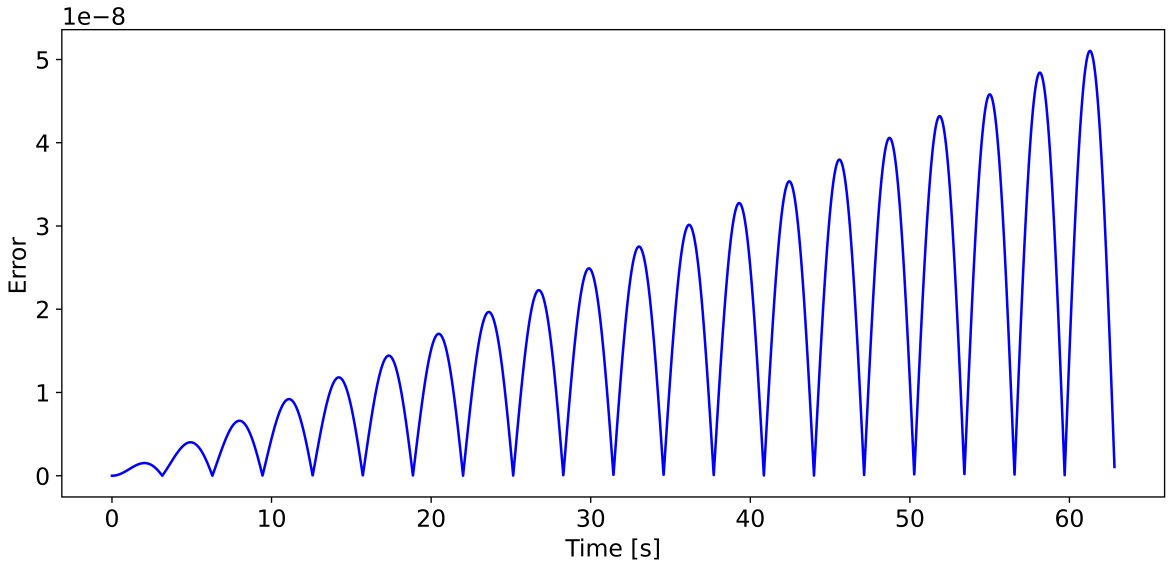

Figure 3: The error for the RK4 method with h = 0.01 is shown for the mass-spring system. The error scales with x10^4 from the previous error in Fig 2 as we'd expect from a truncation error O(h^4)

magnitude of the pendulum equation at all specific points of θ and ω such that it points in the direction of flow of the system, with the length of each arrow representing the strength of the field at that point. Plotting the vector field with the pendulum equations for specific $\beta = 0.25$, 1, 2, 3 we find that indeed our RK4 solver accurately represents the damped pendulum (see Fig 4), perfectly representing critical damping at our calculated value of 2.

We then choose the specific $\beta = 0.5$ case (a very known result taken from John R Taylor's Classical Mechanics [7]) and introduce a range of driving amplitudes A = 0.9, 1.07, 1.47 and 1.5 to shift our study to that of the damped driven pendulum. From our theory we would expect the system to present in some form of steady Period-1 orbits, and collapse towards the accumulation point as A increases. This is perfectly captured by Fig 7 which shows the dynamic shift from Period-1 orbits to Period-n orbits approaching the accumulation point. As this is the case we are able to comfortably move on to assess the main research question for this paper; namely we turn our attention toward the topology of the chaos produced by this system at A = 1.5.

3.4 Poincaré Sections

Following the theory, we first discuss the methods behind plotting our Poincaré section. As we discussed before, this system is stroboscopic meaning we can set our $\tau(\underline{X}_k) = \frac{2\pi}{\omega_D}$. We need to collate a sample of these periods so now have our change in time $dt = \frac{2\pi}{\omega_D \cdot \text{steps}}$ where our steps are a varying parameter of our choosing. We then define our time array as running from our initial time, 0, to our final time $\frac{2\pi N}{\omega_D}$ in steps of dt. N here is also

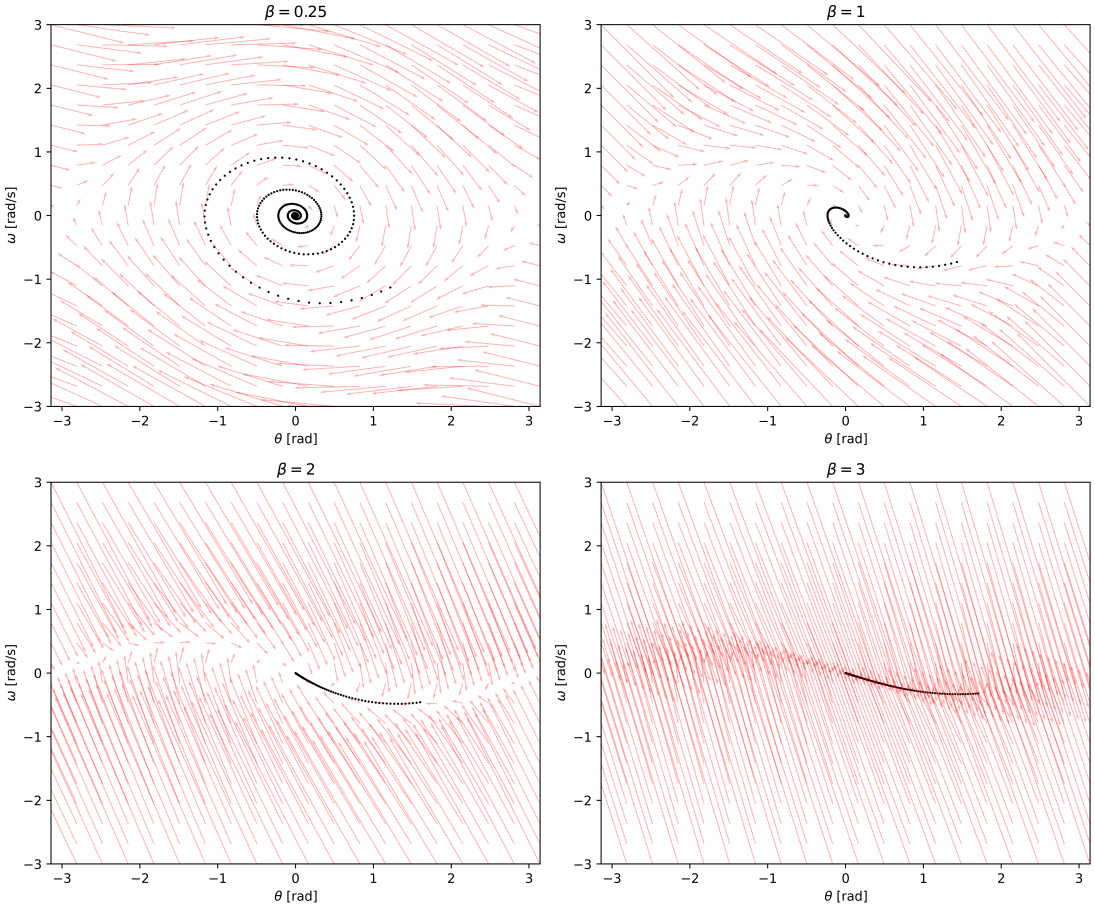

Figure 4: Phase space of the damped pendulum for varying β. For values of 0.25 and 1, underdamping is clearly shown by the spiraling of the trajectory towards the point [0,0] from the initial value $\theta_0 = \frac{1}{4}\pi$ rad. For $\beta = 3$ overdamping is observed as the phase lines of the trajectory compete against each other. At $\beta_c = 2$, all phase lines of the system collapse nicely into the point [0,0] as is expected of critical damping.

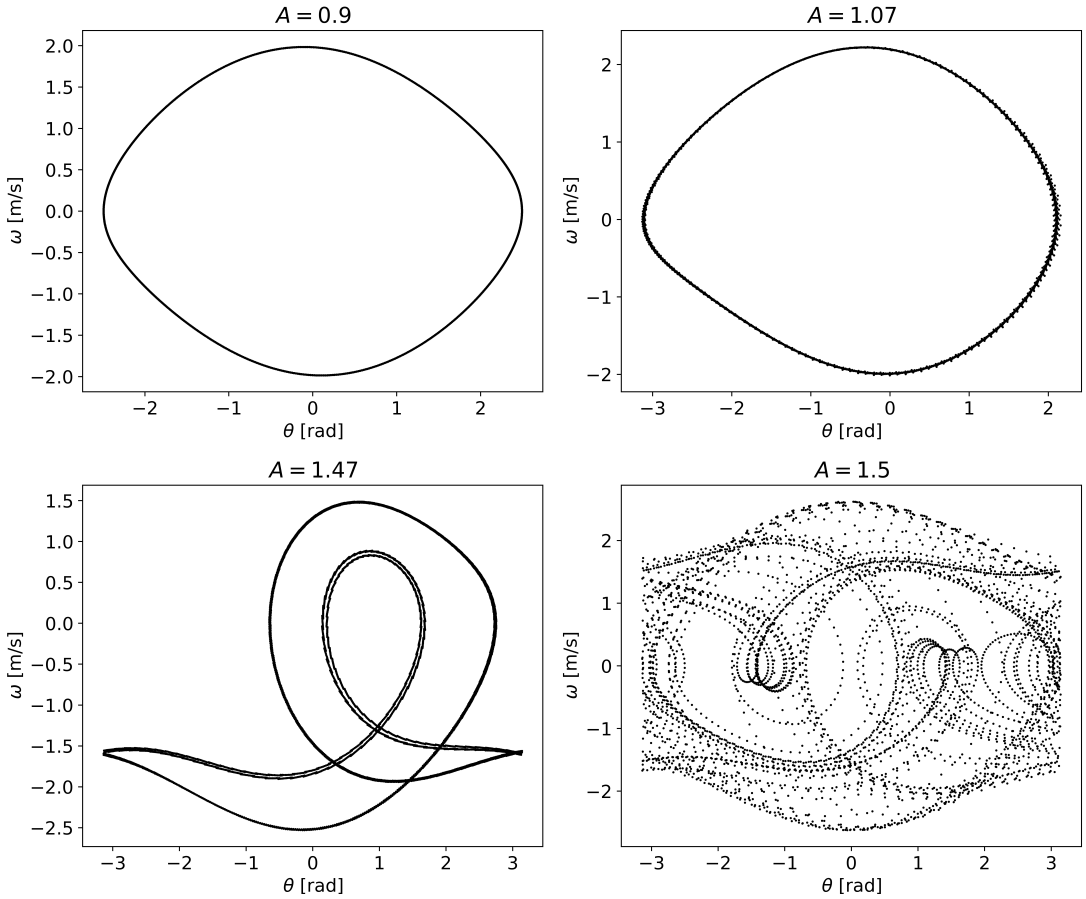

Figure 5: An analysis on the stability of the damped driven pendulum by considering the change in phase space of the system as the driving amplitude is increased from 0.9 to 1.5. Transience is skipped by assuming a start time of 100s. The system portrays the exact behaviour we would expect from our theory section, with Period-1 orbits at low A, and an accumulation point at high A. This means that we can begin to explore the topology of the system without the worry that the RK4 is not dynamic enough to estimate our chaotic regions of the pendulum.

a parameter of our choosing and represents the number of cycles we want the system to make, namely the iterations of the algorithm. For this article we will create three plots, each to represent one of a low, moderate and high resolution Poincaré section. The variables for each case are shown in Table 1. An additional variable skips is added for the purpose of skipping the transient phases of the simulation. For example a skip of 100 with a step of 100 would result in the first 10,000 points being ignored. This is done so our focus can be explicitly on the dynamics of the chaos and not on how this dynamic is reached. The choice is made to keep the skip the same for all of the systems in order to allow the computational analysis to be the most accurate measure it can be.

Table 1: The variables used in each resolution of the Poincaré section are shown. All use the same skip to ensure that the computational analysis is as fair as possible. Increasing the number of iterations and steps increases the resolution of the system.

	Low	**Moderate**	**High**
Iterations N	25,000	50,000	100,000
Steps	25	100	250
Skip	100	100	100

We then turn to plotting the bifurcation. To assess the extent to which the RK4 solver can be used to simulate chaos, the bifurcation plot is probably the most important of all the plots in this paper. It works by essentially looping through Poincaré plots with N = 1,000, steps = 300 and skip = 100 to extract the specific omega values for amplitude values ranging 1 to 1.65 and then plots the specific 'A' value with all of its present omega values. If our phase plots in Fig 5 were correct as we predict they are, the bifurcation plot should show period doubling up until some accumulation point in the region of A = 1.5, while we should also only see one 'path' present for A = 1 due to the Period-1 orbit of this system. The bifurcation plot should also show period doubling accelerating to the accumulation point as our theory predicts. We plot this for 1000 simulations to ensure the full range of motion is captured for each driving amplitude.

3.5 Topological Analysis

Topological Folding

We can finally now start to assess the topology of the system. We begin with the idea of 'kneading bread' as we attempt to visualise the foldings in our system. Following the topological folding and unstable periodic orbits sections of our theory we attempt to develop a computational method to analyse the RK4s ability to plot these regions. This actually turns out to be quite simple as the Poincaré plot does most of the heavy lifting for us. We define a θ_n which is the original angle and a θ_{n+1} that is the next angle plotted in our set of angles from our Poincaré sections (we choose the use of the moderate resolution

diagram for time sake as it allows modifications to be easily made to the system. The end discussion of this paper will attempt to justify our approach to this). We then define that for each θ_n the next in the set of θ_n will be θ_{n+1} and follow the same logic for our set of θ_{n+1}. We then simply plot our set of θ_n against θ_{n+1} and attempt to visualise the folding of the system as predicted by the Poincaré map, $P(X_n)$ in our theory.

Topological Skeleton

Next we wish to visualise this in a way that allows us to see that the energy of the system is not spiraling to infinity. As such we choose our line equation $\theta_{n+1} = \theta_n$ to correctly sample the Period-1 orbits. We use the numerical solver of Newton-Raphson to predict two exact angles and stroboscopic velocities to start at, and then implement the RK4 solver to integrate these forward in time. From our theory, we predict that if our RK4 is correctly able to analyse chaos we should see these to be wrapped at the box dimensions, and periodically loop around each other in a braid-like manner.

3.6 Fidelity Analysis

Coverage

A large portion of our end discussion will be focused on the fidelity of the systems produced by the low, moderate and high resolution Poincaré sections. We start our method by choosing a section around the strange attractor. This will be shown in the results but for now we discuss the area of $\theta \in [1.3, 1.8]$ rad, $\omega \in [-0.4, 0.3]$ rad/s. We then divide this space into a 20x20 grid and count what percentage of these cells are occupied. This tells us how much of the phase space is occupied for each resolution. We expect these to be very similar if not the same.

Correlation Dimension

To calculate the correlation dimension, we then take a random sample of up to 5000 points for each resolution. We calculate the distance between pairs of points and count pairs at different radius thresholds (running as log states of -2 to 20) as the Grassberger-Procaccia algorithm states. By fitting a log-log slope of the fraction of point pairs separated by the radius (the correlation integral), and the radius itself, the correlation dimension for each resolution of the Poincaré section is calculated. As we are running our driving amplitude at quite a high level in comparison to the chaotic region around this value (as we should see in the results), we expect quite a high correlation dimension of approximately 0.8 (save for the 2 space dimensions).

Lyapunov Exponent Estimate

Finalising our fidelity analysis we must produce a method for estimating the largest Lyapunov exponent. We take a similar first step to that of the correlation dimension of which we find the neighbours of each point in our area and compute the log distance between these. We then calculated the average divergence rate across all the pairs which gave us the largest Lyapunov exponent. This is considered quite a fast response in the world of chaotic dynamics as we are only considering 100 base points and 20 sequential points (2000 samples total), however this is not of major concern to us as our result will depend on how each resolution of Poincaré section relates to the next, not in how precise our Lyapunov exponent is to the true value for the system.

3.7 Computational Performance

The last section of the methodology to mention is the computational analysis. While shorter than the other sections this shouldn't be thought of as being less important. With project deadlines and human error, trying to run scripts that take decades to realise one thing has been plotted incorrectly and then having to run it all over again is incredibly impractical. The method we used to assess how the resolution of the Poincaré section affected the computational efficiency of the RK4 numerical solver was by importing the modules time (more specifically perf_counter) and psutil. Inserting these around the RK4 implementation in the Poincaré sections allowed us to track the time each took to run, and the memory usage of each. We used the import json to save these results and plot them together in the results section. This analysis was performed on an apple M1 silicone core and therefore results may differ from the ones we calculated depending on the system used. The module number was also not implemented.

4 Results

This section of the paper presents the findings of our analysis. We diverge from the theory and methods to present the figures and data we generated from these sections. Figure 5 shows the Poincaré plot generated by the moderate resolution parameters on the left side. There is a clear strange attractor in the bottom right, identified in the plot on the right. The dimensions of this strange attractor were used for the fidelity analysis; $\theta \in [1.3, 1.8]$ rad, $\omega \in [-0.4, 0.3]$ rad/s covers this dimension best.

4.1 Poincaré Section & Bifurcation Diagram

The bifurcation is then plotted in Figure 6 by imagining sweeping through the Poincaré sections for A = 1.0 to 1.65 in increments of 0.001 and plotting the stroboscopic velocities

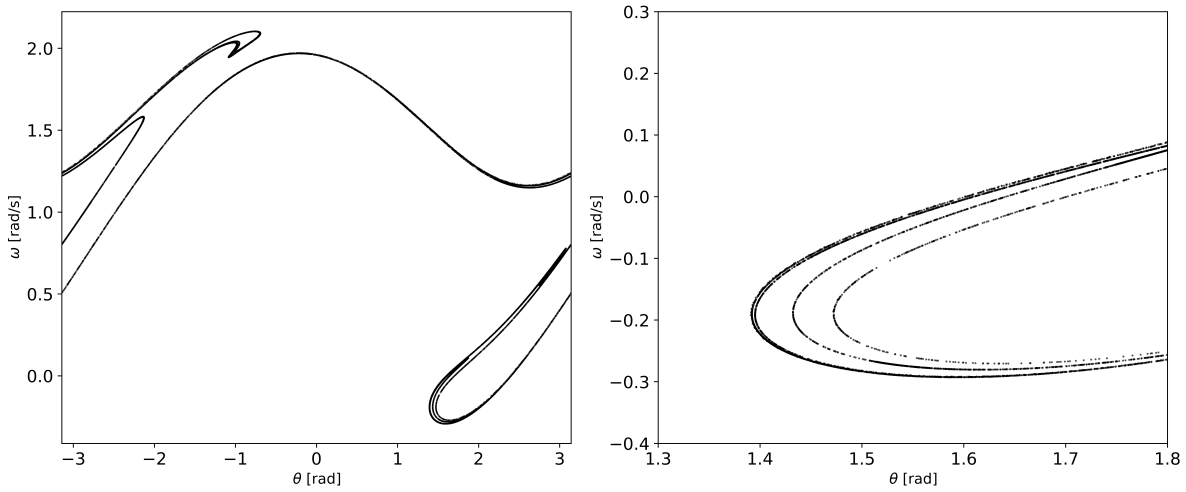

Figure 6: The plot on the left shows the Poincaré section of the system at A = 1.5, $\beta = 0.5$, $\omega_D = 0.67$, N = 50,000, skips = 100, steps = 100. All initial conditions are zero for this system such that chaos emerges from the driving amplitude. A clear strange attractor is seen in the bottom right which is enhanced in the plot on the right. The values here are what gave rise to our decision of box dimensions in our topological analysis. We can see $\theta \in [1.3, 1.8]$ rad, $\omega \in [-0.4, 0.3]$ rad/s covers this dimension best. Beyond this from the Poincaré section we can see that the strange attractor is less well defined.

associated with these. The number of iterations is limited to 1000 from the previous 50,000 in the last figure such that we can instead plot this number of simulations (for our amplitudes) without losing information about the plot as a result of tight clustering. Period-1 orbits can be seen where Fig 5 predicts (at A = 1.0) and period doubling can be seen starting at A = 1.07; accumulating at A = 1.15 (and at A = 1.45→1.5).

4.2 Topology

The result for the topological folding then follows from the Poincaré section in Fig 6, such that Figure 7 shows the current state plotted against the next state. The gradients plotted are sharp and the wrapping of the diffeomorphism, $P(D) \cap D$, is clearly shown. The Smale horseshoe is the amalgamation of an infinite number of the gradients shown.

4.3 Skeleton

The topological braiding then follows from the result in Fig 8. While Figure 9 may look like a messy box of spaghetti it actually shows our incredibly important topological feature of braiding. Two unstable (exact) periodic orbits are plotted against each other for the different initial conditions $A \to \theta_0 = -1.2291,\ \omega_0 = -1.0664$ and $B \to \theta_0 = 1.2587,\ \omega_0 = 1.0452$. A perfect example of this is at (0,-3,25) where the exact orbit A loops and braids around the exact orbit B. Wrapping is still present as we'd expect for the conservation of energy.

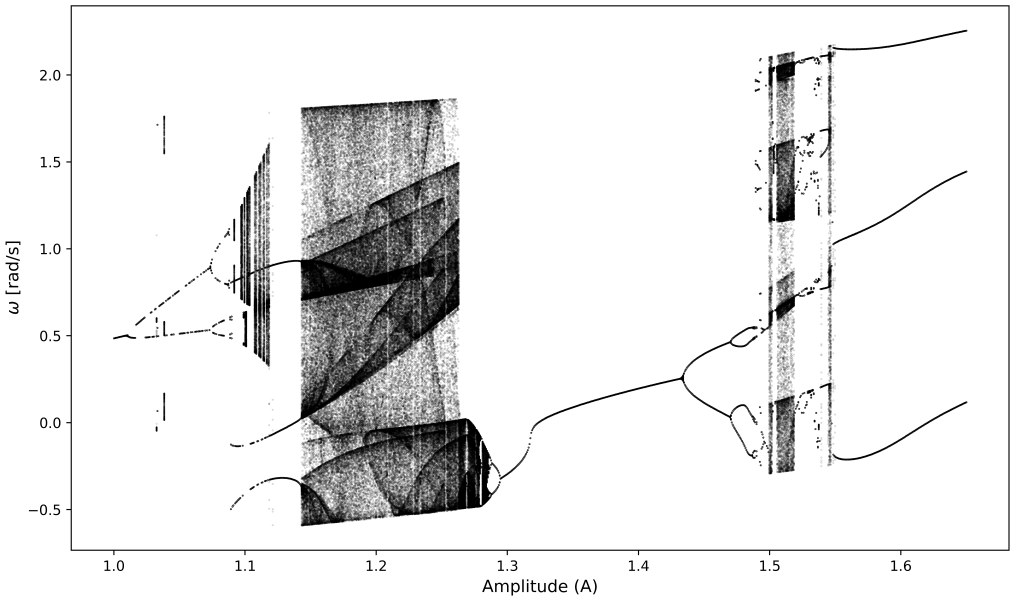

Figure 7: Amplitudes, in a system which follows the parameters laid out in Fig 5 with the exceptions of steps = 300 and N = 1000, swept between 1.0 and 1.65 are plotted in increasing increments of 0.001 against each associated stroboscopic velocity. Period doubling starting at A = 1.07, as Fig 5 suggests, can be seen until an accumulation point at $\sim$ A = 1.15. A second accumulation point is found at A = 1.5, again as Fig 5 suggests.

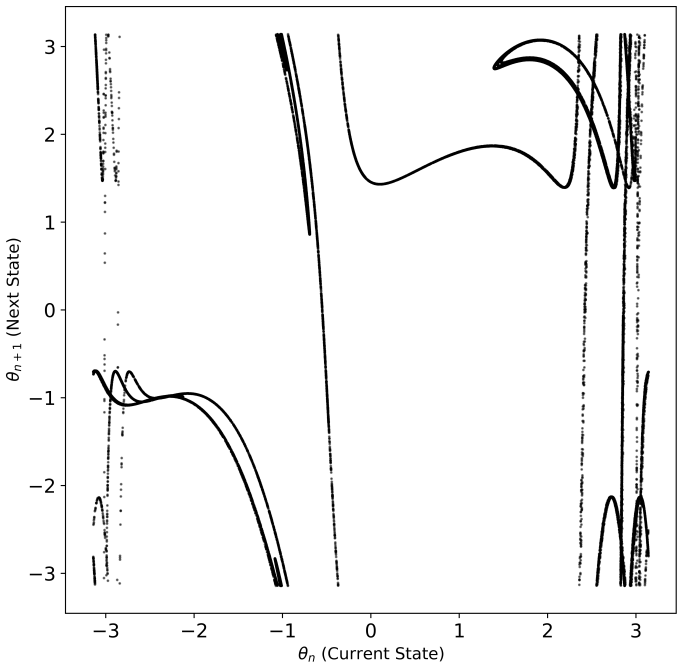

Figure 8: The current state of the Poincaré is plotted against the next. The foldings are clearly wrapped as we would expect for this analysis and The gradients are sharp and well defined. The system is calculated using the theta values in the Poincaré section in Fig 6.

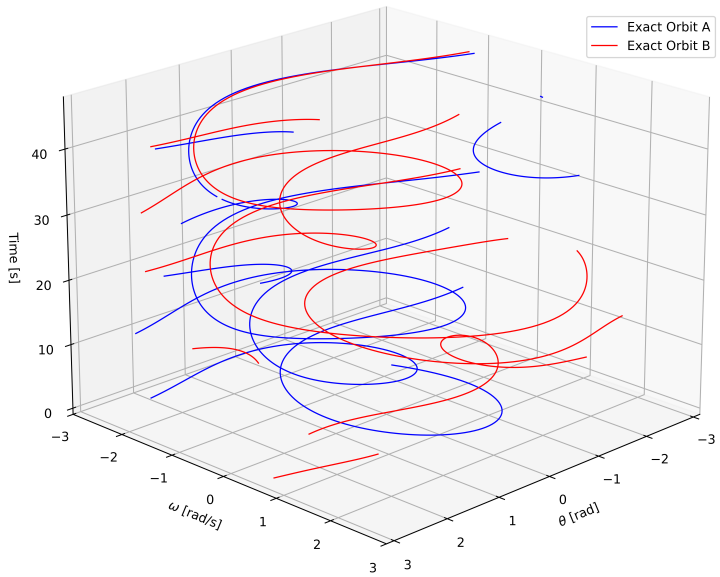

Figure 9: Two unstable periodic orbits for slightly different starting values of $A \rightarrow \theta_0 = -1.2291,\ \omega_0 = -1.0664$ and $B \rightarrow \theta_0 = 1.2587,\ \omega_0 = 1.0452$. A clear example of the braiding we expect can be seen at (0,-3,25) where the exact orbit A loops around and under the exact orbit B. We can see that as the system integrates through time the wrapping is maintained at the box boundaries to cap the energy of the system.

4.4 Fidelity Results

We can then plot our fidelity analysis results in Table 2. These results show us that all resolutions of the RK4 are acceptable for plotting chaos as is given by the positive Lyapunov exponent for these. They also tell us that by plotting higher resolutions, we achieve more accurate measures of the chaos (low-high resolution difference in Lyapunov exponent of +0.1415) but that our conservation of correlation dimension remains relatively the same (1.3% difference between low-moderate and moderate-high), and that our coverage does not vary by much (0.6% difference between low-moderate/high).

Table 2: The fidelity results of the system are shown. The high resolution results demonstrate a stronger prediction of chaos in their Lyapunov estimate and the correlation dimension and coverage remains relatively constant for all the resolutions.

	Low	**Moderate**	**High**
Total Points	24,900	49,900	99,900
Correlation Dimension	0.791	0.802	0.824
Lyapunov Estimate λ_1	0.3240	0.4262	0.4655
Coverage (%)	22.8	22.2	22.2

4.5 Computational Results

Our last result is the results from the computational analysis. Table 3 shows the important values of total steps and points/sec as well as integration time, and peak memory. The results tell us that for higher resolutions we expect longer run times (39x greater runtime for high resolution than low) although the unique result that lower resolutions introduce higher peak memory is also identified. This point will be returned to in the discussion section.

Table 3: The computational results are shown. The higher resolution plots demonstrate a larger total runtime, processing at rates slower than that of the low resolution section. An interesting result emerges from the peak memory which will be discussed in the discussion section.

	Periods	**Steps**	**Total Steps**	**Time (s)**	**Peak Mem. (MB)**	**Points/s**
Low	25,000	25	625,000	12.53	74.84	1986.48
Moderate	50,000	100	5,000,000	100.17	27.86	498.16
High	100,000	250	25,000,000	492.63	27.94	202.79

5 Discussion

5.1 Topology and RK4

To assess the validity of using the RK4 method for topological analysis, we must first confirm its ability to capture baseline chaotic dynamics. We see that we can very accurately depict a visual representation of the strange attractor using the RK4 at a moderate resolution of h = 0.01. We can then see in Fig 7 that the regions that we predicted to be most chaotic displayed this feature (see A = 1.5) and that the regions we predicted to be Period-1 orbits also showed this (see A = 1.0). With this we can certainly validate that the basic chaotic features of the system are being portrayed and as such at the most basic level the RK4 solver is an effective way of capturing the dynamics of the system.

Now assessing the topological folding and braiding of the system we can see that sticking to the h = 0.01 RK4 solver, Fig 8 shows our method effectively captures the folding of the system, with the wrapping being present as you trace the path from any one point to another. Once again, this shows that the theoretical prediction of our map, $P(D) \cap D$, accurately representing a diffeomorphism that stops the energy of the system spiralling infinitely is correctly achieved when using a moderate resolution RK4. Fig 9 also shows that we can calculate the Period-1 orbits when using Newton-Raphson to predict some starting co-ordinates and integrating them through time using the RK4 solver. These orbits perfectly braid around each other as we discussed in the methods section. Considering the results shown in Figs 5-8 we can say with a high level of assurance that the RK4 solver can be used to plot the basic chaotic properties needed for a Poincaré

section and Bifurcation plot, as well as the more complex properties needed for the folding and braiding. This was all done using only a moderate resolution and so we now must discuss the fidelity of the system and raise the question of how much of an improvement to the accuracy of the system does decreasing the step size of the RK4 solver make?

To answer this we consider the results of Table 2. As the results section mentions, this table shows the final result of our fidelity analysis for this system. Typically we look for a higher Lyapunov exponent being a result of more well defined chaos, and a lower coverage percentage for the same system being a good thing as this determines how well defined the points are which we plotted. A larger coverage would have been caused by a larger truncation error. The correlation dimension requires slightly more context as we need to assess how well conserved this value is for the system rather than mentioning the explicitly calculated value.

From Table 2 we can see a noticeable difference between the calculated values of each of these at the different resolutions. It is clear that the moderate resolution section shows stronger support for chaos than the low resolution section as is evidenced by its higher produced estimate of the Lyapunov exponent (+0.1022) and lower coverage (-0.6%). The conservation of the correlation dimension is given as 98.63% (rounded to 4sf). This is a very good conservation value between the two resolutions for the considerable difference between the step-size of the two. It shows that although the truncation does make a difference, it is mainly only seen in the prediction of the Lyapunov exponent and not in the correlation dimension.

If instead we now consider the differences in the moderate and high resolution cases we find the high case does give stronger estimates (+0.0393 for the Lyapunov exponent) of chaos, we see the interesting result which is that the difference is quite a lot smaller than that of the low-moderate case (about 2.6x to be exact) and that there is no difference in the coverage. The conservation is given as 97.33% which is actually worse than that of the low-moderate case. This is not too significant as the conservation is still very high, however as there is such a difference in the growth of the convergence of the Lyapunov estimates this will become a very core part of the conclusion.

5.2 Computational Analysis

Now stepping outside of the chaotic dynamics of the system, we instead discuss the computational results. To do this we must use the results from Table 3 to paint a picture of how each system responded to the implementation of the RK4 solver. We see that, as we expect, plotting a lower resolution graph of lower total steps causes a much faster integration time than that of a higher resolution graph. For an initial eight-fold increase in total steps (low-moderate resolution), we see a decrease of 3.99x in the total steps plotted

per second, whereas for the forty-fold increase in steps (low-high resolution), we see a decrease of 9.80x in the total steps plotted per second. This result leads to the prediction that plotting larger and larger numbers of points causes the system to slow at some rate proportional to the number of points plotted. We refrain from calculating this constant due to the lack of data we have for this, however this response is still very important in listing the behaviour of the RK4 solver and will be integrated into the conclusion.

An interesting result from Table 3 is the peak memory for each resolution. We expect a harsher strain on the processor for larger total steps but don't actually see this result. The reduction in peak memory usage is approximately 2.67x less in the high resolution plot (27.94MB) than the low resolution plot (74.84MB) but remains roughly the same for the moderate (27.86MB) and high plots. This discrepancy is not a feature of the RK4 algorithm itself, which theoretically scales linearly in memory with the number of array elements. Rather, it is an artifact of the software's memory management environment; highly rapid executions (such as the 12-second low-resolution run) often report higher transient memory spikes before the environment can optimize overhead. Because this result reaches some cut-off point and then remains constant we can conclude the integration time as being the sole restriction for plotting smaller steps of the RK4 solver.

5.3 Comparison of RK4 to Other Work

Comparing the RK4 solver to other work is quite a straightforward task. We have already shown that the RK4 solver is a measure of accuracy higher than the Euler method in the theory of this paper, and instead can switch our attention to the Bulirsch-Stoer Numerical Solver. Similar to Runge's method of averaging the slope between two points in the Euler method, the Bulirsch-Stoer solver uses Richardson extrapolation and the modified midpoint method to take one large step and divide it into many smaller steps, which it then extrapolates as $h \to 0$.

This is a far more precise measure as we have already seen that decreasing step size increases the precision of the RK4 solver, and the Bulirsch-Stoer solver takes this approximation to 0 effectively eliminating truncation error (but introducing much smaller hardware error and rounding error). As this paper attempts to cover a small niche in the large field of chaos theory, applying the Bulirsch-Stoer solver in this way has not typically been done, however many papers do exist comparing methods of numerical solver. Foundational papers such as 'Numerical Recipes' [8] praise the Bulirsch-Stoer solver for being "the best-known way to obtain high accuracy solutions to ordinary differential equations with minimal computational effort," however this only applies for extreme precision for very smooth functions and one caveat listed to this method is non-smooth equations where it is better to "go back to Runge-Kutta with an adaptive stepsize choice. That method does an excellent job of feeling its way through rocky or discontinuous terrain." Because

the Poincaré sections in this study require exact stroboscopic sampling at fixed intervals, the RK4 method's fixed step size allows for direct, perfectly spaced data extraction while the Bulirsch-Stoer method would require computationally expensive interpolation to achieve the same fixed-interval mapping, negating its efficiency advantages.

We can therefore conclude that for the damped driven pendulum, while applying the Bulirsch-Stoer solver may produce incredibly accurate measures of chaos, the computational efficiency of applying the method would far outweigh the benefits of using the solver in comparison to the RK4. We could, however, choose to use the solver on explicit calculations for small variables that we may need a measure for, such as a better Lyapunov calculation than the one we performed.

6 Conclusion

This study set out to answer a primary research question: to what extent is the RK4 method a valid numerical approach for the topological assessment of the chaotic driven damped pendulum? We now have all the pieces of the puzzle and are left to put everything together.

We can confidently say that the RK4 solver excels at plotting the baseline of chaotic systems. The generated Poincaré section and bifurcation in Figs 6 and 7 are precise and accurate depictions of what we set out to achieve in the theory and methods sections of this paper. Not only that, the method also exceeds expectations in plotting the topology of chaotic systems. The plots generated in Figs 8 and 9 are exactly what we assumed we'd see from our theory. As mentioned in the results and discussion, we see that the folding and braiding are correctly visualised for the results we'd assumed to see in this analysis.

Adding to this, we see that by increasing our resolution of the Poincaré section plotted, our estimate for the maximum Lyapunov exponent grows increasingly more accurate. This means that for future research involving these calculations, there is more room to improve the accuracy using small steps and larger amounts of iterations with the RK4 solver.

However, we must also bring the topic of computational analysis into this conclusion. To run greater precision of the RK4, we need to increase our runtime proportional to the measure of the increase in resolution from the previous. This raises the question of at what point is the increase in accuracy made to the exponent calculated less valuable than the time it takes to perform the calculation? We can see from our results and discussion that for our low-high resolution results, we gained an increase in convergence of 31% but at the cost of a 39 fold increase to the runtime of the section. We can also see that the overall increase to convergence actually decreases as resolution increases. Between

the low and moderate resolution sections there was an increase in convergence of 24% while between the moderate and high resolution there was an increase in convergence of 8.4%. This decrease is especially concerning when addressing the increase to the resolution as between the low and moderate sections there is an eight fold increase to resolution, while between the moderate and high sections there is a five fold increase to resolution. Therefore we infer that there is a proportional relationship between higher resolutions and the decrease in the growth of convergence. We can then assess that therefore for higher resolution plots the RK4 method is not a computationally acceptable method of generating highly accurate points. We could instead consider the use of the Bulirsch-Stoer solver from the previous section.

Future Work

There is a lot of necessity for future work with this project and similar projects on computational methods and chaos. This project was very limited by the amount of computational results we produced. By plotting our results in the style of Figure 10, we are able to actually see a non-linear fit begin to emerge. It would be preferable in future work to plot a variety of resolutions ranging from an ultra low plot of total steps $\sim$ 100,000 to an ultra high resolution plot of total steps $\sim$ 100,000,000. We could then use these results to assess the curve of runtime against the number of points extracted and potentially view this non-linearity.

We could also use this result to assess at which point the growth rate of the convergence of the Lyapunov exponent decreases by plotting the calculated estimates for the exponents against the runtime of the resolutions used and then calculating a local gradient at separate points on the curve. We could improve this measure as well by attempting to implement the Bulirsch-Stoer method for some small amounts of these plots.

Another area to explore is introducing more topological fidelity to the analysis. This project was limited in that we had one measure for the 'strength' of the chaos and one measure for the conservation of the chaos across different RK4 resolutions, but we could improve this by introducing more dimensions. 'The Dimension of Chaotic Attractors' [5] provides a fantastic analysis of how we would perform these calculations. We would choose to introduce the Lyapunov dimension. We would be able to use the Kaplan-Yorke equation to show where the system stops expanding and starts contracting. Our results would hopefully follow that of our Lyapunov exponent results in that the result would increase until a critical resolution wherein it would converge for all resolutions higher than that.

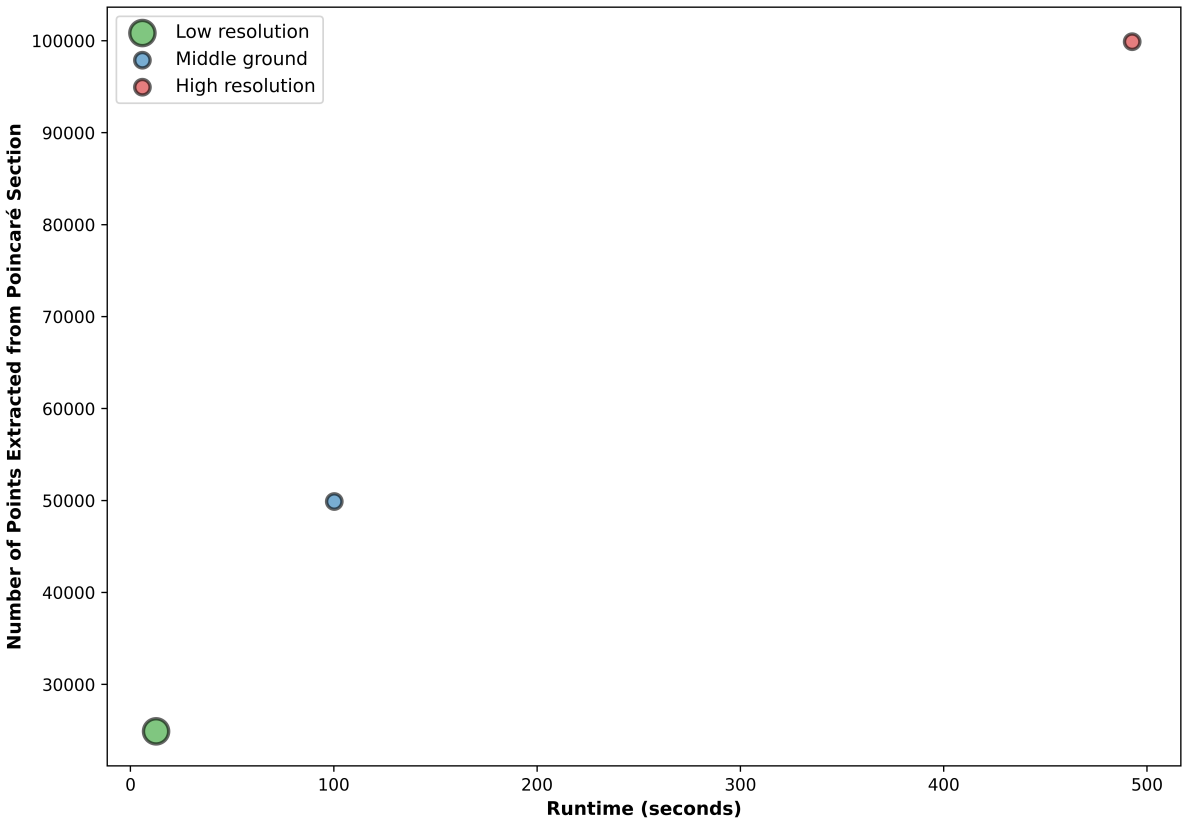

Figure 10: A visual analysis of Table 3. Different resolutions of the RK4 solver are shown plotting runtime against points extracted from the Poincaré section. A non-linear fit is starting to emerge and could become better defined for a larger number of resolutions.

Acknowledgements

We'd like to acknowledge the use of Google's Gemini and Anthropic's Claude. Both AI helped develop small amounts of code to integrate all of the features of the fidelity analysis together, and helped track the performance of each script. This was built off of a very well defined template but helped to speed up the development of the code. The only code affected were small sections of parts 6 and 7 of the code found in the appendix.

References

[1] H. Poincaré, *Sur le problème des trois corps et les équations de la dynamique*, Acta Mathematica **13**, 1-270 (1890).

[2] E. N. Lorenz, *Deterministic Nonperiodic Flow*, Journal of the Atmospheric Sciences **20**, 130-141 (1963).

[3] J. C. Butcher, *A History of Runge-Kutta Methods*, Applied Numerical Mathematics **20**(3), 247-260 (1996).

[4] P. Grassberger and I. Procaccia, *Characterisation of Strange Attractors*, Physical Review Letters **50**(5), 346-349 (1983).

[5] J. D. Farmer, E. Ott, and J. A. Yorke, *The Dimension of Chaotic Attractors*, Physica D **7**, 153-180 (1983).

[6] J. M. Norman, *Huygens Invents the Pendulum Clock, Increasing Accuracy Sixty Fold*, HistoryofInformation.com (2025). `https://www.historyofinformation.com/detail.php?entryid=3506`

[7] J. R. Taylor, *Classical Mechanics* (University Science Books, Sausalito, CA, 2005).

[8] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes: The Art of Scientific Computing*, 3rd ed. (Cambridge University Press, 2007).

A Python Code

The complete Python master script is reproduced below. It is structured into nine sections executed in order: (1) benchmarks, (2) phase-space plots, (3-5) Poincaré sections at three resolutions, (6) bifurcation diagram, (7) topological braid, (8) fidelity analysis, and (9) computational performance.

```
"""
Master Compilation: Pendulum Dynamics, Chaos, and Topology

Order of Execution:
```

```
1. Benchmarks
2. Phase space plots
3. Poincare plots
4. Bifurcation plots
5. Topology plots
6. Topology analysis
7. Computational analysis
"""

import sys
import os
import csv
import json
import psutil
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import animation
from time import perf_counter
from pathlib import Path
from scipy.spatial.distance import cdist
from scipy.special import comb
from mpl_toolkits.mplot3d import Axes3D

# UTILITY AND CORE ODE FUNCTIONS

def RK4(x, y, f, h):
    """Standard 4th-Order Runge-Kutta Numerical Integrator."""
    k1 = h * f(x, y)
    k2 = h * f(x + 0.5 * h, y + 0.5 * k1)
    k3 = h * f(x + 0.5 * h, y + 0.5 * k2)
    k4 = h * f(x + h, y + k3)
    return y + (k1 + 2 * k2 + 2 * k3 + k4) / 6

def Mass_on_spring(t, x_v):
    """ODE for a simple mass on a spring (Benchmark)."""
    x, v = x_v
    k = 1
    m = 1
    dxdt = x_v[1]
    dvdt = -(k / m) * x_v[0]
    return np.array([dxdt, dvdt])

def DampedPendulum(t, theta_w, B):
    """ODE for a Damped Pendulum."""
    theta, w = theta_w
    l = 9.81
    g = 9.81
```

```
    dthetadt = theta_w[1]
    dwdt = -g/l * np.sin(theta_w[0]) - B * theta_w[1]
    return np.array([dthetadt, dwdt])

def DampedODE_vector_field(theta_val, w_val, B):
    """Vector field generator for the Damped Pendulum."""
    l = 9.81
    g = 9.81
    dthetadt = w_val
    dwdt = -g/l * np.sin(theta_val) - B * w_val
    return dthetadt, dwdt

def damped_pendulum(initial_theta, beta_val):
    """Integration loop for Damped Pendulum."""
    t0 = 0
    tend = 500
    h = 0.1
    n = int((tend - t0) / h)
    theta_i = np.zeros(n)
    w_i = np.zeros(n)
    t = np.zeros(n)
    t[0] = t0
    theta_i[0] = initial_theta
    w_i[0] = 0
    for j in range(n-1):
        x_v_current = np.array([theta_i[j], w_i[j]])
        x_v_next = RK4(t[j], x_v_current,
            lambda current_t, current_theta_w:
                DampedPendulum(current_t, current_theta_w, beta_val), h)
        theta_i[j+1] = x_v_next[0]
        w_i[j+1] = x_v_next[1]
        t[j+1] = t[j] + h
        if theta_i[j+1] > np.pi:
            theta_i[j+1] -= 2*np.pi
        if theta_i[j+1] < -np.pi:
            theta_i[j+1] += 2*np.pi
    return t, w_i, theta_i

def DrivenDampedPendulum(t, theta_w, B, A):
    """ODE for a Driven Damped Pendulum."""
    theta, w = theta_w
    l = 9.81
    g = 9.81
    Wd = 0.67
    dthetadt = theta_w[1]
    dwdt = -(g/l) * np.sin(theta_w[0]) - B * theta_w[1] + A * np.cos(Wd
       * t)
```

```
    return np.array([dthetadt, dwdt])

def driven_damped_pendulum(N0_inputs, initial_omega, A):
    """Integration loop for Driven Damped Pendulum."""
    all_omega = []
    all_theta = []
    omega0 = np.zeros(N0_inputs)
    for i in range(N0_inputs):
        omega0[i] = (i**2 + initial_omega) * 0.3
    t0 = 0
    theta0 = 0
    tend = 500
    h = 0.1
    n = int((tend - t0) / h)
    theta_i = np.zeros(n)
    w_i = np.zeros(n)
    t = np.zeros(n)
    t[0] = t0
    theta_i[0] = theta0
    w_i[0] = omega0[i]
    for j in range(n-1):
        x_v_current = np.array([theta_i[j], w_i[j]])
        x_v_next = RK4(t[j], x_v_current,
            lambda current_t, current_theta_w:
                DrivenDampedPendulum(current_t, current_theta_w, 0.5, A)
                    , h)
        theta_i[j+1] = x_v_next[0]
        w_i[j+1] = x_v_next[1]
        t[j+1] = t[j] + h
    theta_wrapped = (theta_i + np.pi) % (2 * np.pi) - np.pi
    all_omega.append(w_i)
    all_theta.append(theta_wrapped)
    return t, all_omega, all_theta, omega0

def DrivenDampedODE_vector_field(theta_val, w_val, B, A, t_val):
    """Vector field generator for Driven Damped Pendulum."""
    l = 9.81
    g = 9.81
    Wd = 0.67
    dthetadt = w_val
    dwdt = -g/l * np.sin(theta_val) - B*w_val + A*np.cos(Wd*t_val)
    return dthetadt, dwdt

# 1. BENCHMARKS

def run_benchmarks():
    """
```

```
    Benchmarking RK4 solver against mass-on-spring analytic solution.
    Compares RK4 numerical solution with known analytic solution.
    """
    print("\n- Running Benchmarks -")
    x0 = 10.0
    v0 = 0.0
    t0 = 0.0
    tend = 20 * np.pi
    h = 0.1
    n = int((tend - t0) / h)
    t = np.zeros(n)
    x = np.zeros(n)
    v = np.zeros(n)
    t[0], x[0], v[0] = t0, x0, v0
    for i in range(n - 1):
        x_v_current = np.array([x[i], v[i]])
        x_v_next = RK4(t[i], x_v_current, Mass_on_spring, h)
        x[i+1] = x_v_next[0]
        v[i+1] = x_v_next[1]
        t[i+1] = t[i] + h
    x_bench = x0 * np.cos(t)
    v_bench = -x0 * np.sin(t)
    error_x = np.abs(x_bench - x)
    plt.rcParams['font.size'] = 14
    fig1, ax1 = plt.subplots(figsize=(10, 6))
    ax1.plot(t, x, color='red', marker='.', markersize=5,
             linestyle='', label='Position (RK4)')
    ax1.plot(t, x_bench, color='pink', linestyle='-',
             linewidth=2, alpha=0.8, label='Position (Bench)')
    ax1.set_xlabel('Time [s]')
    ax1.set_ylabel('Position [m]', color='k')
    ax1.tick_params(axis='y', labelcolor='k')
    ax1.set_ylim(-14, 14)
    ax2 = ax1.twinx()
    ax2.plot(t, v, color='black', marker='.', markersize=5,
             linestyle='', label='Velocity (RK4)')
    ax2.plot(t, v_bench, color='gray', linestyle='-',
             linewidth=2, alpha=0.8, label='Velocity (Bench)')
    ax2.set_ylabel('Velocity [m/s]', color='black')
    ax2.tick_params(axis='y', labelcolor='black')
    ax2.set_ylim(-14, 14)
    lines1, labels1 = ax1.get_legend_handles_labels()
    lines2, labels2 = ax2.get_legend_handles_labels()
    ax1.legend(lines1 + lines2, labels1 + labels2,
               loc='upper right', fontsize=6)
    fig1.tight_layout()
    fig1.savefig('Benchmarking_Rk4.png', dpi=1400)
```

```
    plt.close(fig1)
    fig2, ax3 = plt.subplots(figsize=(10, 5))
    ax3.plot(t, error_x, 'b', label="Absolute Error in Position (x)")
    ax3.set_xlabel('Time [s]')
    ax3.set_ylabel('Error')
    fig2.tight_layout()
    fig2.savefig('Benchmarking_Rk4_Error.png', dpi=1400)
    plt.close(fig2)

# 2. PHASE SPACE PLOTS

def run_phase_space_plots():
    """
    Damped and Driven Damped Pendulum Phase Plots.
    Visualizes phase space behavior and time-dependent vector fields.
    """
    print("\n- Running Phase Space Plots -")
    betas = [0.25, 1, 2, 3]
    theta_start = 90
    fig, axs = plt.subplots(2, 2, figsize=(12, 10))
    axs = axs.flatten()
    for idx, beta in enumerate(betas):
        t_vals, omega_vals, theta_vals = damped_pendulum(theta_start,
            beta)
        axs[idx].scatter(theta_vals, omega_vals, s=0.5, color='k')
        x_min, x_max = -np.pi, np.pi
        y_min, y_max = -3, 3
        x_grid = np.linspace(x_min, x_max, 20)
        v_grid = np.linspace(y_min, y_max, 20)
        X_grid, V_grid = np.meshgrid(x_grid, v_grid)
        U_grid, W_grid = DampedODE_vector_field(X_grid, V_grid, beta)
        axs[idx].quiver(X_grid, V_grid, U_grid, W_grid,
                        color='r', alpha=0.3, scale=20, width=0.002)
        axs[idx].set_title(f'($\\beta = {beta}$)')
        axs[idx].set_xlabel('$\\theta$ [rad]')
        axs[idx].set_ylabel('$\\omega$ [rad/s]')
        axs[idx].set_xlim(x_min, x_max)
        axs[idx].set_ylim(y_min, y_max)
    plt.tight_layout()
    plt.savefig('Phaseplot_damped_wide.png', dpi=700)
    plt.close()
    As = [0.9, 1.07, 1.47, 1.5]
    fig, axs = plt.subplots(2, 2, figsize=(12, 10))
    axs = axs.flatten()
    for idx, A in enumerate(As):
        t_vals, omega_vals, theta_vals, omega0 = \
            driven_damped_pendulum(1, 0, A)
```

```
        t_vals = np.array(t_vals).flatten()
        omega_vals = np.array(omega_vals).flatten()
        theta_vals = np.array(theta_vals).flatten()
        mask = t_vals > 100
        if np.any(mask):
            axs[idx].scatter(theta_vals[mask], omega_vals[mask],
                             s=0.5, color='k')
        axs[idx].set_title(f'$A = {A}$')
        axs[idx].set_xlabel('$\\theta$ [rad]')
        axs[idx].set_ylabel('$\\omega$ [m/s]')
    plt.tight_layout()
    plt.savefig('Phaseplot_driven_damped.png', dpi=700)
    plt.close()
    x_min, x_max = -3, 3
    y_min, y_max = -3, 3
    x_grid = np.linspace(x_min, x_max, 20)
    v_grid = np.linspace(y_min, y_max, 20)
    X_grid, V_grid = np.meshgrid(x_grid, v_grid)
    parameters = [(0.25, 0.9), (0.25, 1.5), (1.0, 0.9), (1.0, 1.5)]
    fig, axs = plt.subplots(2, 2, figsize=(12, 10))
    axs = axs.flatten()
    quivers = []
    for idx, (beta, A) in enumerate(parameters):
        ax = axs[idx]
        U, W = DrivenDampedODE_vector_field(X_grid, V_grid, beta, A, 0)
        Q = ax.quiver(X_grid, V_grid, U, W,
                      color='r', alpha=0.3, scale=20, width=0.002)
        quivers.append(Q)
        ax.set_title(f"$\\beta={beta}$, $A={A}$ | Time: 0.00 s")
        ax.set_xlabel('$\\theta$ [rad]')
        ax.set_ylabel('$\\omega$ [m/s]')
        ax.set_xlim(x_min, x_max)
        ax.set_ylim(y_min, y_max)
    def update_quiver(t):
        for idx, (beta, A) in enumerate(parameters):
            ax = axs[idx]
            Q = quivers[idx]
            U_new, W_new = DrivenDampedODE_vector_field(
                X_grid, V_grid, beta, A, t)
            Q.set_UVC(U_new, W_new)
            ax.set_title(
                f"$\\beta={beta}$, $A={A}$ | Time: {t:.2f} s")
        return quivers
    t_frames = np.linspace(0, 20, 50)
    anim = animation.FuncAnimation(fig, update_quiver,
                                   frames=t_frames, interval=100,
                                   blit=False)
```

```
    plt.rcParams['animation.embed_limit'] = 500.0
    anim.save('phase_portraits.gif', writer='pillow', fps=10)
    plt.close(fig)

# 3. POINCARE PLOTS

def run_poincare_low_res():
    """
    Low Resolution Poincare Section.
    Fast computation with 25 steps per period for quick visualization.
    """
    print("\n- Running Low Resolution Poincare -")
    process = psutil.Process(os.getpid())
    A_val = 1.5
    omega_d = 0.67
    beta_val = 0.5
    N = 25000
    steps = 25
    skip = 100
    dt = 2 * np.pi / (omega_d * steps)
    time_arr = np.arange(0.0, N * 2 * np.pi / omega_d, dt)
    nsteps = len(time_arr)
    omega = np.zeros(nsteps)
    theta = np.zeros(nsteps)
    integration_start = perf_counter()
    for j in range(nsteps-1):
        x_v_current = np.array([theta[j], omega[j]])
        x_v_next = RK4(time_arr[j], x_v_current,
            lambda t, yw: DrivenDampedPendulum(t, yw, beta_val, A_val),
               dt)
        theta[j+1] = x_v_next[0]
        omega[j+1] = x_v_next[1]
    integration_time = perf_counter() - integration_start
    peak_memory = process.memory_info().rss / (1024**2)
    offset = steps // 4
    start_idx = skip * steps + offset
    theta_p = theta[start_idx :: steps]
    omega_p = omega[start_idx :: steps]
    theta_p_wrapped = (theta_p + np.pi) % (2 * np.pi) - np.pi
    performance_data = {
        'script': '13_poincare_low_resolution.py',
        'periods': N,
        'steps_per_period': steps,
        'total_steps': nsteps,
        'points_extracted': len(theta_p),
        'integration_time_sec': integration_time,
        'peak_memory_mb': peak_memory,
```

```
        'points_per_second': len(theta_p) / integration_time
            if integration_time > 0 else 0
    }
    with open('performance_13.json', 'w') as f:
        json.dump(performance_data, f, indent=2)
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 6))
    ax1.scatter(theta_p_wrapped, omega_p, s=0.1,
                color='black', alpha=0.5)
    ax1.set_xlabel('$\\theta$ [rad]', fontsize=12)
    ax1.set_ylabel('$\\omega$ [rad/s]', fontsize=12)
    ax1.set_xlim(-np.pi, np.pi)
    ax2.scatter(theta_p_wrapped, omega_p, s=0.5, color='k', alpha=0.6)
    ax2.set_xlabel('$\\theta$ [rad]', fontsize=12)
    ax2.set_ylabel('$\\omega$ [rad/s]', fontsize=12)
    ax2.set_xlim(1.3, 1.8)
    ax2.set_ylim(-0.4, 0.3)
    plt.tight_layout()
    plt.savefig('poincare_plot_low_def.pdf', dpi=700)
    plt.close()

def run_poincare_reg_res():
    """
    Regular Poincare Section Analysis - Chaotic Attractors.
    Shows stroboscopic sampling at driving frequency.
    """
    print("\n- Running Regular Poincare -")
    process = psutil.Process(os.getpid())
    A_val = 1.5
    omega_d = 0.67
    beta_val = 0.5
    N = 50000
    steps = 100
    skip = 100
    dt = 2 * np.pi / (omega_d * steps)
    t_array = np.arange(0.0, N * 2 * np.pi / omega_d, dt)
    nsteps = len(t_array)
    omega = np.zeros(nsteps)
    theta = np.zeros(nsteps)
    integration_start = perf_counter()
    for j in range(nsteps-1):
        x_v_current = np.array([theta[j], omega[j]])
        x_v_next = RK4(t_array[j], x_v_current,
            lambda t, yw: DrivenDampedPendulum(t, yw, beta_val, A_val),
               dt)
        theta[j+1] = x_v_next[0]
        omega[j+1] = x_v_next[1]
    integration_time = perf_counter() - integration_start
```

```
    peak_memory = process.memory_info().rss / (1024**2)
    offset = steps // 4
    start_idx = skip * steps + offset
    theta_p = theta[start_idx :: steps]
    omega_p = omega[start_idx :: steps]
    theta_p_wrapped = (theta_p + np.pi) % (2 * np.pi) - np.pi
    performance_data = {
        'script': '09_poincare_section.py',
        'periods': N,
        'steps_per_period': steps,
        'total_steps': nsteps,
        'points_extracted': len(theta_p),
        'integration_time_sec': integration_time,
        'peak_memory_mb': peak_memory,
        'points_per_second': len(theta_p) / integration_time
            if integration_time > 0 else 0
    }
    with open('performance_09.json', 'w') as f:
        json.dump(performance_data, f, indent=2)
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 6))
    ax1.scatter(theta_p_wrapped, omega_p, s=0.1,
                color='black', alpha=0.5)
    ax1.set_xlabel('$\\theta$ [rad]', fontsize=12)
    ax1.set_ylabel('$\\omega$ [rad/s]', fontsize=12)
    ax1.set_xlim(-np.pi, np.pi)
    ax2.scatter(theta_p_wrapped, omega_p, s=0.5, color='k', alpha=0.6)
    ax2.set_xlabel('$\\theta$ [rad]', fontsize=12)
    ax2.set_ylabel('$\\omega$ [rad/s]', fontsize=12)
    ax2.set_xlim(1.3, 1.8)
    ax2.set_ylim(-0.4, 0.3)
    plt.tight_layout()
    plt.savefig('poincare_plot.png', dpi=700)
    plt.close()
    theta_n = theta_p_wrapped[:-1]
    theta_n_plus_1 = theta_p_wrapped[1:]
    plt.figure(figsize=(8, 8))
    plt.scatter(theta_n, theta_n_plus_1, s=1.0, color='k', alpha=0.5)
    plt.xlabel('$\\theta_n$ (Current State)', fontsize=12)
    plt.ylabel('$\\theta_{n+1}$ (Next State)', fontsize=12)
    plt.savefig('Topological_folding.png', dpi=700)
    plt.close()

def run_poincare_super_res():
    """
    Super-High Resolution Poincare Section.
    100k cycles x 250 steps per period = comprehensive attractor
        resolution.
```

```
    """
    print("\n- Running Super-High Resolution Poincare -")
    process = psutil.Process(os.getpid())
    A_val = 1.5
    omega_d = 0.67
    beta_val = 0.5
    N = 100000
    steps = 250
    skip = 100
    dt = 2 * np.pi / (omega_d * steps)
    time_arr = np.arange(0, N * 2 * np.pi / omega_d, dt)
    nsteps = len(time_arr)
    omega = np.zeros(nsteps)
    theta = np.zeros(nsteps)
    integration_start = perf_counter()
    for j in range(nsteps-1):
        x_v_current = np.array([theta[j], omega[j]])
        x_v_next = RK4(time_arr[j], x_v_current,
            lambda t, y: DrivenDampedPendulum(t, y, beta_val, A_val), dt
               )
        theta[j+1] = x_v_next[0]
        omega[j+1] = x_v_next[1]
    integration_time = perf_counter() - integration_start
    peak_memory = process.memory_info().rss / (1024**2)
    theta = theta[skip*steps:]
    omega = omega[skip*steps:]
    offset = steps // 4
    theta_poincare = theta[offset :: steps]
    omega_poincare = omega[offset :: steps]
    theta_poincare = (theta_poincare + np.pi) % (2*np.pi) - np.pi
    num_points = len(theta_poincare)
    performance_data = {
        'script': '14_poincare_super_resolution.py',
        'periods': N,
        'steps_per_period': steps,
        'total_steps': nsteps,
        'points_extracted': num_points,
        'integration_time_sec': integration_time,
        'peak_memory_mb': peak_memory,
        'points_per_second': num_points / integration_time
            if integration_time > 0 else 0
    }
    with open('performance_14.json', 'w') as f:
        json.dump(performance_data, f, indent=2)
    fig, axes = plt.subplots(2, 2, figsize=(16, 12))
    ax = axes[0, 0]
    ax.scatter(theta_poincare, omega_poincare, s=0.5,
```

```
                color='black', alpha=0.6)
    ax.set_xlim(-np.pi, np.pi)
    focus_mask = ((theta_poincare >= 1.3) & (theta_poincare <= 1.8) &
                  (omega_poincare >= -0.4) & (omega_poincare <= 0.3))
    ax = axes[0, 1]
    ax.scatter(theta_poincare[focus_mask], omega_poincare[focus_mask],
                s=2, color='red', alpha=0.7)
    ax.scatter(theta_poincare[~focus_mask], omega_poincare[~focus_mask],
                s=0.1, color='black', alpha=0.3)
    ax.set_xlim(1.3, 1.8)
    ax.set_ylim(-0.4, 0.3)
    ultra_zoom_mask = ((theta_poincare >= 1.45) & (theta_poincare <=
        1.65) &
                       (omega_poincare >= -0.2) & (omega_poincare <=
                          0.1))
    ax = axes[1, 0]
    ax.scatter(theta_poincare[ultra_zoom_mask],
                omega_poincare[ultra_zoom_mask],
                s=3, color='darkred', alpha=0.8)
    ax.set_xlim(1.45, 1.65)
    ax.set_ylim(-0.2, 0.1)
    ax = axes[1, 1]
    theta_n = theta_poincare[:-1]
    theta_n_plus_1 = theta_poincare[1:]
    ax.scatter(theta_n, theta_n_plus_1, s=0.5,
                color='darkblue', alpha=0.5)
    ax.set_xlim(-np.pi, np.pi)
    ax.set_ylim(-np.pi, np.pi)
    plt.tight_layout()
    plt.savefig('poincare_super_resolution.pdf', dpi=400,
                bbox_inches='tight')
    plt.close()

# 4. BIFURCATION PLOTS

def run_bifurcation_diagram():
    """
    Bifurcation Diagram - Route to Chaos.
    Shows how system behavior changes with driving amplitude.
    """
    print("\n- Running Bifurcation Diagram -")
    omega_d = 0.67
    beta_val = 0.5
    A_vals = np.linspace(1, 1.65, 1000)
    steps = 300
    skip = 100
    N_cycles = 1000
```

```
    dt = 2 * np.pi / (omega_d * steps)
    nsteps = N_cycles * steps
    time_arr = np.arange(0, nsteps * dt, dt)
    A_plot = []
    omega_plot = []
    for idx, A in enumerate(A_vals):
        theta = np.zeros(nsteps)
        omega = np.zeros(nsteps)
        for j in range(nsteps-1):
            x_v_current = np.array([theta[j], omega[j]])
            x_v_next = RK4(time_arr[j], x_v_current,
                lambda t, yw: DrivenDampedPendulum(t, yw, beta_val, A),
                    dt)
            theta[j+1] = x_v_next[0]
            omega[j+1] = x_v_next[1]
        offset = steps // 4
        omega_p = omega[skip * steps + offset :: steps]
        for om_p in omega_p:
            A_plot.append(A)
            omega_plot.append(om_p)
    plt.figure(figsize=(12, 7))
    plt.scatter(A_plot, omega_plot, s=0.05, color='black', alpha=0.3)
    plt.xlabel('Amplitude (A)', fontsize=12)
    plt.ylabel('$\\omega$ [rad/s]', fontsize=12)
    plt.savefig('Bifurcation_diagram.png', dpi=700)
    plt.close()

# 5. TOPOLOGY PLOTS

def run_topological_braid():
    """
    Topological Skeleton - Braiding of Unstable Periodic Orbits.
    3D visualization of chaotic braiding structures.
    """
    print("\n- Running Topological Braid -")
    A_val = 1.5
    omega_d = 0.67
    beta_val = 0.5
    T = 2 * np.pi / omega_d
    p1_th_exact = -1.2291
    p1_om_exact = -1.0664
    p2_th_exact =  1.2587
    p2_om_exact =  1.0452

    def integrate_exact_path(th0, om0, num_periods,
                             steps_per_period=200):
        dt = T / steps_per_period
```

```
        n_steps = num_periods * steps_per_period
        th_path = np.zeros(n_steps)
        om_path = np.zeros(n_steps)
        t_path  = np.zeros(n_steps)
        th, om = th0, om0
        for i in range(n_steps):
            t = i * dt
            th_wrapped = (th + np.pi) % (2 * np.pi) - np.pi
            th_path[i] = th_wrapped
            om_path[i] = om
            t_path[i]  = t
            k1_th = om * dt
            k1_om = (-np.sin(th) - beta_val*om
                     + A_val*np.cos(omega_d*t)) * dt
            th2 = th + 0.5*k1_th; om2 = om + 0.5*k1_om
            k2_th = om2 * dt
            k2_om = (-np.sin(th2) - beta_val*om2
                     + A_val*np.cos(omega_d*(t+0.5*dt))) * dt
            th3 = th + 0.5*k2_th; om3 = om + 0.5*k2_om
            k3_th = om3 * dt
            k3_om = (-np.sin(th3) - beta_val*om3
                     + A_val*np.cos(omega_d*(t+0.5*dt))) * dt
            th4 = th + k3_th; om4 = om + k3_om
            k4_th = om4 * dt
            k4_om = (-np.sin(th4) - beta_val*om4
                     + A_val*np.cos(omega_d*(t+dt))) * dt
            th += (k1_th + 2*k2_th + 2*k3_th + k4_th) / 6.0
            om += (k1_om + 2*k2_om + 2*k3_om + k4_om) / 6.0
        return th_path, om_path, t_path

    num_periods_to_integrate = 5
    th_a, om_a, t_a = integrate_exact_path(
        p1_th_exact, p1_om_exact, num_periods_to_integrate)
    th_b, om_b, t_b = integrate_exact_path(
        p2_th_exact, p2_om_exact, num_periods_to_integrate)

    def hide_wrapping_lines(th, om, t):
        jumps = np.where(np.abs(np.diff(th)) > np.pi)[0] + 1
        th_clean = np.insert(th, jumps, np.nan)
        om_clean = np.insert(om, jumps, np.nan)
        t_clean  = np.insert(t,  jumps, np.nan)
        return th_clean, om_clean, t_clean

    th_a_c, om_a_c, t_a_c = hide_wrapping_lines(th_a, om_a, t_a)
    th_b_c, om_b_c, t_b_c = hide_wrapping_lines(th_b, om_b, t_b)
    fig = plt.figure(figsize=(10, 8))
    ax = fig.add_subplot(111, projection='3d')
```

```
    ax.plot(th_b_c, om_b_c, t_b_c, color='red',
            linewidth=1, label='Exact Orbit B')
    ax.set_xlabel('$\\theta$ [rad]')
    ax.set_ylabel('$\\omega$ [rad/s]')
    ax.set_zlabel('Time [s]')
    ax.set_xlim(-np.pi, np.pi)
    ax.set_ylim(-3, 3)
    ax.view_init(elev=20, azim=45)
    plt.savefig('Topological_skeleton.png', dpi=700)
    plt.close()

# 6. TOPOLOGY ANALYSIS

class DualLogger(object):
    """Custom logger for fidelity analysis output."""
    def __init__(self, filename="poincare_analysis_results.txt"):
        self.terminal = sys.stdout
        self.log = open(filename, "w", encoding="utf-8")
    def write(self, message):
        self.terminal.write(message)
        self.log.write(message)
    def flush(self):
        self.terminal.flush()
        self.log.flush()

def extract_poincare_points(N, steps, skip, A_val, omega_d, beta_val):
    dt = 2 * np.pi / (omega_d * steps)
    time_arr = np.arange(0, N * 2 * np.pi / omega_d, dt)
    nsteps = len(time_arr)
    omega = np.zeros(nsteps)
    theta = np.zeros(nsteps)
    for j in range(nsteps - 1):
        x_v_current = np.array([theta[j], omega[j]])
        x_v_next = RK4(time_arr[j], x_v_current,
            lambda t, y: DrivenDampedPendulum(t, y, beta_val, A_val), dt
               )
        theta[j+1] = x_v_next[0]
        omega[j+1] = x_v_next[1]
    theta_clean = theta[skip*steps:]
    omega_clean = omega[skip*steps:]
    offset = steps // 4
    theta_p = theta_clean[offset :: steps]
    omega_p = omega_clean[offset :: steps]
    theta_p_wrapped = (theta_p + np.pi) % (2 * np.pi) - np.pi
    return theta_p_wrapped, omega_p, len(theta_p)

def count_points_in_region(theta, omega, theta_range, omega_range):
```

```
    mask = ((theta >= theta_range[0]) & (theta <= theta_range[1]) &
            (omega >= omega_range[0]) & (omega <= omega_range[1]))
    return np.sum(mask)

def compute_point_density(theta, omega, grid_size=20):
    theta_edges = np.linspace(-np.pi, np.pi, grid_size + 1)
    omega_min, omega_max = omega.min(), omega.max()
    omega_edges = np.linspace(omega_min, omega_max, grid_size + 1)
    density = np.zeros((grid_size, grid_size))
    for i in range(grid_size):
        for j in range(grid_size):
            count = count_points_in_region(
                theta, omega,
                (theta_edges[i], theta_edges[i+1]),
                (omega_edges[j], omega_edges[j+1]))
            density[i, j] = count
    return density, theta_edges, omega_edges

def estimate_correlation_dimension(theta, omega, max_samples=5000):
    if len(theta) > max_samples:
        idx = np.random.choice(len(theta), max_samples, replace=False)
        theta_sample, omega_sample = theta[idx], omega[idx]
    else:
        theta_sample, omega_sample = theta, omega
    points = np.column_stack([theta_sample, omega_sample])
    distances = cdist(points, points)
    r_values = np.logspace(-2, 0, 20)
    C_r = []
    for r in r_values:
        count = np.sum(distances < r) - len(points)
        C_r.append(max(count, 1))
    C_r = np.array(C_r)
    valid_idx = C_r > 0
    log_r = np.log(r_values[valid_idx])
    log_C = np.log(C_r[valid_idx])
    coeffs = np.polyfit(log_r, log_C, 1)
    return coeffs[0], r_values, C_r

def detect_clusters(theta, omega, radius=0.1):
    points = np.column_stack([theta, omega])
    n = len(points)
    if n == 0:
        return 0, []
    visited = np.zeros(n, dtype=bool)
    clusters = []
    for i in range(n):
        if visited[i]:
```

```
            continue
        cluster = [i]
        visited[i] = True
        queue = [i]
        while queue:
            current = queue.pop(0)
            distances = np.linalg.norm(
                points - points[current], axis=1)
            neighbors = np.where(
                (distances < radius) & ~visited)[0]
            for j in neighbors:
                visited[j] = True
                cluster.append(j)
                queue.append(j)
        clusters.append(cluster)
    return len(clusters), clusters

def compute_lyapunov_exponent_estimate(theta, omega,
                                       max_samples=2000):
    if len(theta) > max_samples:
        idx = np.random.choice(len(theta), max_samples,
                               replace=False)
        theta_sample, omega_sample = theta[idx], omega[idx]
    else:
        theta_sample, omega_sample = theta, omega
    points = np.column_stack([theta_sample, omega_sample])
    divergences = []
    for i in range(min(100, len(points) - 10)):
        for j in range(i+1, min(i+20, len(points))):
            dist_now = np.linalg.norm(points[i] - points[j])
            if dist_now > 1e-6:
                divergences.append(np.log(dist_now))
    return np.mean(divergences) if divergences else 0.0

def analyze_coverage(theta, omega, grid_size=20):
    density, _, _ = compute_point_density(theta, omega, grid_size)
    non_empty_cells = np.sum(density > 0)
    total_cells = grid_size * grid_size
    return 100 * non_empty_cells / total_cells, density

def run_fidelity_analysis():
    print("\n- Running Fidelity Analysis -")
    original_stdout = sys.stdout
    sys.stdout = DualLogger("poincare_analysis_results.txt")
    print("\n" + "="*70)
    print("POINCARE SECTION TOPOLOGICAL FIDELITY ANALYSIS")
    print("="*70)
```

```
    A_val = 1.5; omega_d = 0.67; beta_val = 0.5
    print("\n[SCRIPT 13] Low Resolution")
    theta_09, omega_09, _ = extract_poincare_points(
        25000, 25, 100, A_val, omega_d, beta_val)
    print(f"Extracted: {len(theta_09)} points")
    print("\n[SCRIPT 09] Regular Resolution")
    theta_12, omega_12, _ = extract_poincare_points(
        50000, 100, 100, A_val, omega_d, beta_val)
    print(f"Extracted: {len(theta_12)} points")
    print("\n[SCRIPT 14] Super-High Resolution")
    theta_14, omega_14, _ = extract_poincare_points(
        100000, 250, 100, A_val, omega_d, beta_val)
    print(f"Extracted: {len(theta_14)} points")
    print("\n" + "="*70 + "\nSPATIAL ANALYSIS\n" + "="*70)
    focus_region = {"theta": (1.3, 1.8), "omega": (-0.4, 0.3)}
    for label, th, om in [("13", theta_09, omega_09),
                          ("09", theta_12, omega_12),
                          ("14", theta_14, omega_14)]:
        c = count_points_in_region(th, om,
            focus_region["theta"], focus_region["omega"])
        print(f"Focus Region Points (Script {label}): {c} "
              f"({100*c/max(len(th),1):.1f}%)")
    cov_09, _ = analyze_coverage(theta_09, omega_09)
    cov_12, _ = analyze_coverage(theta_12, omega_12)
    cov_14, _ = analyze_coverage(theta_14, omega_14)
    print(f"\nCoverage: Script 13 ({cov_09:.1f}%), "
          f"Script 09 ({cov_12:.1f}%), Script 14 ({cov_14:.1f}%)")
    print("\n" + "="*70 + "\nTOPOLOGICAL PROPERTIES\n" + "="*70)
    D2_09, _, _ = estimate_correlation_dimension(theta_09, omega_09)
    D2_12, _, _ = estimate_correlation_dimension(theta_12, omega_12)
    D2_14, _, _ = estimate_correlation_dimension(theta_14, omega_14)
    print(f"Correlation Dimension (D_2): "
          f"Script 13 ({D2_09:.3f}), "
          f"Script 09 ({D2_12:.3f}), "
          f"Script 14 ({D2_14:.3f})")
    lyap_09 = compute_lyapunov_exponent_estimate(theta_09, omega_09)
    lyap_12 = compute_lyapunov_exponent_estimate(theta_12, omega_12)
    lyap_14 = compute_lyapunov_exponent_estimate(theta_14, omega_14)
    print(f"Lyapunov Est (lambda): "
          f"Script 13 ({lyap_09:.4f}), "
          f"Script 09 ({lyap_12:.4f}), "
          f"Script 14 ({lyap_14:.4f})")
    sys.stdout = original_stdout
    print("Full fidelity output written to "
          "'poincare_analysis_results.txt'")

# 7. COMPUTATIONAL ANALYSIS
```

```
793
794 def run_performance_analysis():
795     """
796     PERFORMANCE ANALYSIS - Computational Cost Assessment.
797     Analyzes runtime, memory usage, and efficiency across all
798     Poincare simulations.
799     """
800     print("\n- Running Performance Analysis -")
801     performance_data = []
802     for json_file in Path('.').rglob('performance_*.json'):
803         with open(json_file, 'r') as f:
804             try:
805                 data = json.load(f)
806                 if isinstance(data, list):
807                     for entry in data:
808                         name = entry.get('script',
809                                          entry.get('name', 'Unknown'))
810                         if '12' not in name and 'Ultra' not in name:
811                             performance_data.append(entry)
812                 else:
813                     name = data.get('script',
814                                     data.get('name', 'Unknown'))
815                     if '12' not in name and 'Ultra' not in name:
816                         performance_data.append(data)
817             except Exception:
818                 pass
819     if not performance_data:
820         print("WARNING: No performance data files found!")
821         return
822     extracted_dict = {}
823     for entry in performance_data:
824         name = entry.get('script', entry.get('name', 'Unknown'))
825         if '13' in name or 'Low' in name:
826             label, order, color = 'Low resolution', 0, '#2ca02c'
827         elif '09' in name or 'Regular' in name:
828             label, order, color = 'Middle ground', 1, '#1f77b4'
829         elif '14' in name or 'Super' in name:
830             label, order, color = 'High resolution', 2, '#d62728'
831         else:
832             continue
833         time_val = entry.get('integration_time_sec',
834                              entry.get('time_sec', 0))
835         mem_val  = entry.get('peak_memory_mb',
836                              entry.get('memory_mb', 0))
837         points   = entry.get('points_extracted',
838                              entry.get('points', 0))
839         extracted_dict[label] = (order, label, time_val,
```

```
                                    mem_val, points, color)
    extracted_data = list(extracted_dict.values())
    extracted_data.sort(key=lambda x: x[0])
    scripts      = [item[1] for item in extracted_data]
    times        = [item[2] for item in extracted_data]
    memories     = [item[3] for item in extracted_data]
    points_list  = [item[4] for item in extracted_data]
    colors_list  = [item[5] for item in extracted_data]
    if scripts:
        fig, ax = plt.subplots(figsize=(10, 7))
        sizes = [mem * 3 for mem in memories]
        for time_val, points, color, label, size in zip(
                times, points_list, colors_list, scripts, sizes):
            ax.scatter(time_val, points, s=size, alpha=0.6,
                       color=color, edgecolors='black',
                       linewidth=2, label=label)
        ax.set_xlabel('Runtime (seconds)',
                      fontsize=11, fontweight='bold')
        ax.set_ylabel('Number of Points Extracted from '
                      'Poincare Section',
                      fontsize=11, fontweight='bold')
        ax.set_title('Computational Cost vs. Data Yield Tradeoff',
                     fontsize=14, fontweight='bold')
        ax.legend(fontsize=11)
        plt.tight_layout()
        plt.savefig('performance_tradeoff.png', dpi=700)
        plt.close()
        print("Saved: performance_tradeoff.png")

# MASTER EXECUTION

if __name__ == "__main__":
    print("Starting Main Execution Pipeline...")
    run_benchmarks()
    run_phase_space_plots()
    run_poincare_low_res()
    run_poincare_reg_res()
    run_poincare_super_res()
    run_bifurcation_diagram()
    run_topological_braid()
    run_fidelity_analysis()
    run_performance_analysis()
    print("\nPipeline execution complete! "
          "All plots, logs, and GIFs generated.")
```