Newton's Second Law and the Nonlinear Pendulum

Elijan Jakob Mastnak
Student ID: 28181157

November 2020

Contents

Assignment

1. Thoroughly analyze the motion of the mathematical pendulum with the initial conditions $x(0) = 1$ and $\dot{x}(0) = 0$. Use a variety of numerical methods—the more the better—to solve the pendulum's equation of motion.

2. Using the analytic solution in terms of elliptic integrals for reference, find a time step for which the numerical solution is accurate to at least three decimal places.

3. Observe how the pendulum's amplitude and energy change over time (e.g. 20 periods) as a result of numerical error.

4. Plot the pendulum's motion in position-momentum phase space.

5. *Optional:* Investigate the resonance curve of a driven, damped mathematical pendulum:
$$\frac{\mathrm{d}^2x}{\mathrm{d}t^2} + \beta\frac{\mathrm{d}x}{\mathrm{d}t} + \sin x = v\cos\omega_0 t$$
where β is the damping coefficient while v and ω_0 are the driving amplitude and frequency. Observe the pendulum's angular displacement and velocity for $\beta = 0.5$, $\omega_0 = \frac{2}{3}$ and $v \in (0.5, 1.5)$. Try to detect hysteresis in the pendulum's resonance curve at large driving amplitudes.

6. *Optional:* Analyze the phase space of the van der Pol oscillator
$$\frac{\mathrm{d}^2x}{\mathrm{d}t^2} - \lambda\frac{\mathrm{d}x}{\mathrm{d}t} + x = v\cos\omega_0 t$$
for the parameter values $\omega_0 = 1$, $v = 10$ and $\lambda = 1$ or $\lambda = 100$.

---

1 Theory

To jump right to the solution, see Section 2.

1.1 Newton's Law in One Dimension

Newton's law describes the motion of a particle of mass m in a force field F via the second-order differential equation

$$m\frac{\mathrm{d}^2x}{\mathrm{d}t^2} = F \qquad \text{or, in 3D,} \qquad m\frac{\mathrm{d}^2\boldsymbol{r}}{\mathrm{d}t^2} = \boldsymbol{F}$$

We solve the equation in numerically by working with the equivalent system of first-order equations

$$m\frac{\mathrm{d}x}{\mathrm{d}t} = p \qquad \text{and} \qquad \frac{\mathrm{d}p}{\mathrm{d}t} = F$$

A unique solution requires two initial conditions, $x(t = 0) \equiv x_0$ and $\dot{x}(t = 0) \equiv v_0$.

1.2 The Nonlinear Simple Pendulum

A simple pendulum is a mass m attached to a light, mass-less rod. Motion takes place in a vertical plane with a single degree of freedom, the pendulum's angular displacement θ from the equilibrium position. Newton's law reads

$$\frac{\mathrm{d}^2\theta}{\mathrm{d}t^2} = -\omega_0^2 \sin\theta, \qquad \omega_0 = \sqrt{\frac{g}{l}}$$

where l is the pendulum's length and g is the gravitational acceleration. The equation is non-linear in θ because of the $\sin\theta$ term. The pendulum's period is

$$T_0 = \frac{4}{\omega_0} K\big(\sin^2\tfrac{\theta_0}{2}\big)$$

where $K(m)$ is a complete elliptical integral of the first kind, implemented in this report with the m (and not m^2) convention

$$K(m) = \int_0^1 \frac{\mathrm{d}z}{\sqrt{(1-z^2)(1-mz^2)}} = \int_0^{\pi/2} \frac{\mathrm{d}u}{\sqrt{1-m\sin^2 u}}$$

1.3 Symplectic Integrators

Symplectic integration methods apply to functions f that are a function of only coordinates and preserve's the system's Hamiltonian. In this report, I used the second-order Verlet method and a fourth-order position-extended Forest-Ruth-like method. For a second-order derivative of the coordinate y given by

$$f(y) = \frac{\mathrm{d}^2 y}{\mathrm{d}t^2}$$

and time points $t_n = t_0 + n \cdot h$ with time step h, the Verlet method gives expressions for y_n and $v_n \equiv \dot{y}(n)$ in the form

$$y_{n+1} = y_n + h v_n + \frac{h^2}{2} f(y_n)$$
$$v_{n+1} = v_n + \frac{h}{2}\big[f(y_n) + f(y_{n+1})\big]$$

2 Initial Analytic Steps

2.1 Pendulum Energy and Equation of Motion

Before beginning the simulation, I needed an expression for the simple pendulum's energy. The energy is simply

$$E = T + V = \frac{ml^2}{2}\dot{\theta}^2 + mgl\,(1-\cos\theta)$$

We find the equation of motion from the Lagrangian $L = T - V$

$$L = T - V = \frac{1}{2} ml^2\dot{\theta}^2 - mgl\,(1-\cos\theta)\,,$$

and the corresponding Lagrange-Euler equations for the coordinates θ and $\dot{\theta}$:

$$\frac{\mathrm{d}}{\mathrm{d}t}\left[\frac{\partial L}{\partial \ddot{\theta}}\right] = ml^2\dot{\theta} \equiv \frac{\partial L}{\partial \theta} = -mgl\sin\theta \implies \ddot{\theta} = -\frac{g}{l}\sin\theta$$

2.1.1 Dimensionless Expressions

I found it most convenient to work in a dimensionless set of units with $l = g = 1$, in which the equation of motion and energy are simply

$$\ddot{\theta} = -\sin\theta \qquad \text{and} \qquad E = \frac{\dot{\theta}^2}{2} + 1 - \cos\theta$$

The pendulum is bound for $E < 2$ (e.g. for $\dot{\theta}_0 = 0$ and $\theta_0 \in (0, \pi)$) and free for $E > 2$ (meaning it will "tip over"). The initial energy is

$$E_0 = \frac{\dot{\theta_0}^2}{2} + 1 - \cos\theta_0 \tag{1}$$

All oscillating systems in this report are solved in dimensionless form.

2.2 Implementing Period and Analytic Solution

I implemented the analytic solution for the simple pendulum's angular displacement with the initial condition $\theta(0) = \theta_0, \dot{\theta}(0) = 0$ according to [1]

$$\theta(t) = 2\arcsin\left\{\sin\frac{\theta_0}{2}\,\mathrm{sn}\left[K\left(\sin^2\frac{\theta_0}{2}\right) - t; \sin^2\frac{\theta_0}{2}\right]\right\}$$

where K is the complete elliptic integral of the first kind and sn is the Jacobi elliptic function $\mathrm{sn}(u; m)$ with argument u and parameter m. Meanwhile, the pendulum's period T_0, again for the initial condition $\theta(0) = \theta_0, \dot{\theta}(0) = 0$, is

$$T_0 = \frac{4}{\omega_0} K\left(\sin\frac{\theta_0}{2}\right)$$

I implemented these expressions in Python using the built-in `ellipk` and `ellipj` from the package `scipy.special`. The angular displacement is

```python
import numpy as np
from scipy.special import ellipk, ellipj
def simple_pendulum_analytic(t, initial_state, w0=1.0):
    """ Returns the simple pendulum's angular displacement as a function of
    ↪  time for the initial condition [x0, 0] """
    k = np.sin(initial_state[0] / 2)
    return 2*np.arcsin(k * ellipj(ellipk(k**2) - w0*t, k**2)[0])
```

while the oscillation period reads

```python
def get_simple_pendulum_period(x0):
    """ Returns the simple pendulum's period for initial angular displacement
    ↪  x0 and angular velocity v0 = 0 """
    k = np.sin(x0/2)
    return 4*ellipk(k**2)
```

2.3 Implementing Differential Equations of State

I implemented the pendulum's differential equation of state, which returns angular velocity and acceleration for a given angular displacement and velocity, in two forms: a time-dependent format for use with generic methods for ODEs and a time-independent format, depending on only the angular displacement, for use with symplectic integrators. The following example Python code blocks give a feel for the difference between the standard ODE and symplectic approaches

```python
def get_simple_pendulum_state_ode(state, t):
    """
    Dimensionless differential equation of motion for a simple (mathematical)
    ↪ pendulum written for compatibility with ODE methods.
    :param state: 2-element [position, velocity] array i.e. state = [x, v]
    :return: 2-element array holding velocity and acceleration i.e. [v, a]
    """
    return_state = np.zeros(np.shape(state))
    return_state[0] = state[1]
    return_state[1] = -np.sin(state[0])
    return return_state  # returns both velocity and acceleration
```

```python
def get_simple_pendulum_state_symp(x):
    """
    Dimensionless differential equation of motion for a simple (mathematical)
    ↪ pendulum written for compatibility with symplectic integrators.
    :param x: the pendulum's angular displacement from equilibrium
    :return: the pendulum's angular acceleration
    """
    return -np.sin(x)  # returns only acceleration!
```

Note that the symplectic method is a function of only the system's coordinates and returns only the acceleration. The ODE method is in general a function of the coordinates, velocities and time, and returns both velocity and acceleration.

2.3.1 Numerical Methods Used in This Report

I tested the following 19 methods for numerically solving the simple pendulum's motion. Source code is included in the attached `numerical_methods_odes.py` file.

Fixed time step explicit methods

- Euler method `euler`
- Heun's method `heun` and midpoint method `rk2`.
- Ralston's 3rd Runge-Kutta method `rk3r` and 3rd order strong stability preserving Runge-Kutta `rk3ssp`
- Classic 4th order Runge-Kutta `rk4` and Ralston's 4th order Runge-Kutta method `rk4r`

Single-step, embedded, adaptive-step Runge-Kutta methods:

- Adaptive 4-5th order Runge-Kutta-Fehlberg method `rkf45`
- Adaptive 4-5th order Cash-Karp method `ck45`

Multistep methods

- 4th order multi-step predictor-corrector Adams-Bashforth-Moulton method `pc4`

Symplectic methods

- 2nd order Verlet method `verlet`
- 4th order position-extended Forest-Ruth-like method `pefrl`.

Built-in methods

- SciPy's `odeint`
- SciPy's `solve_ivp` with the explicit Runge-Kutta methods `RK23`, `RK45`, `DOP853`
- SciPy's `solve_ivp` with the implicit `Randau` (family of implicit Runge-Kutta methods) and `BDF` (family of backward-differentiation methods)
- SciPy's `solve_ivp` with the `LSODA` option, a wrapper for FORTRAN code.

2.4 A Few Solution Graphs

It feels appropriate to start with a solution for the pendulum's motion: Figures 1 and 2 show the simple pendulum's motion for three classes of initial conditions. Note the approximately sinusoidal behavior for the moderate initial displacement $\theta_0 = 1.0$, the non-harmonic behavior near the tipping point with $\theta_0 = 0.99\pi$, and the free motion for $\theta_0 = 0.99\pi$ and $\dot{\theta}_0 = 0.1$.

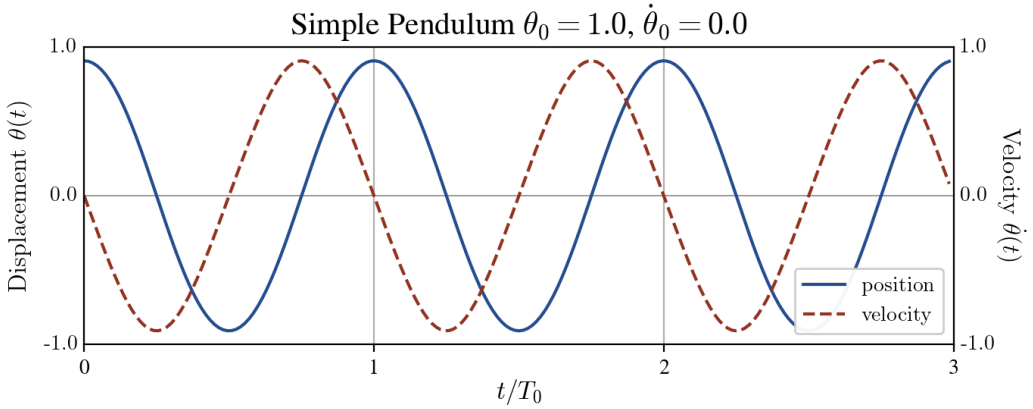

Figure 1: The simple pendulum's approximately harmonic motion for initial conditions $(\theta_0, \dot{\theta}_0) = (1.0, 0)$. Note that both angular displacement θ and angular velocity $\dot{\theta}$ share a single plot. Found with `pefrl`.

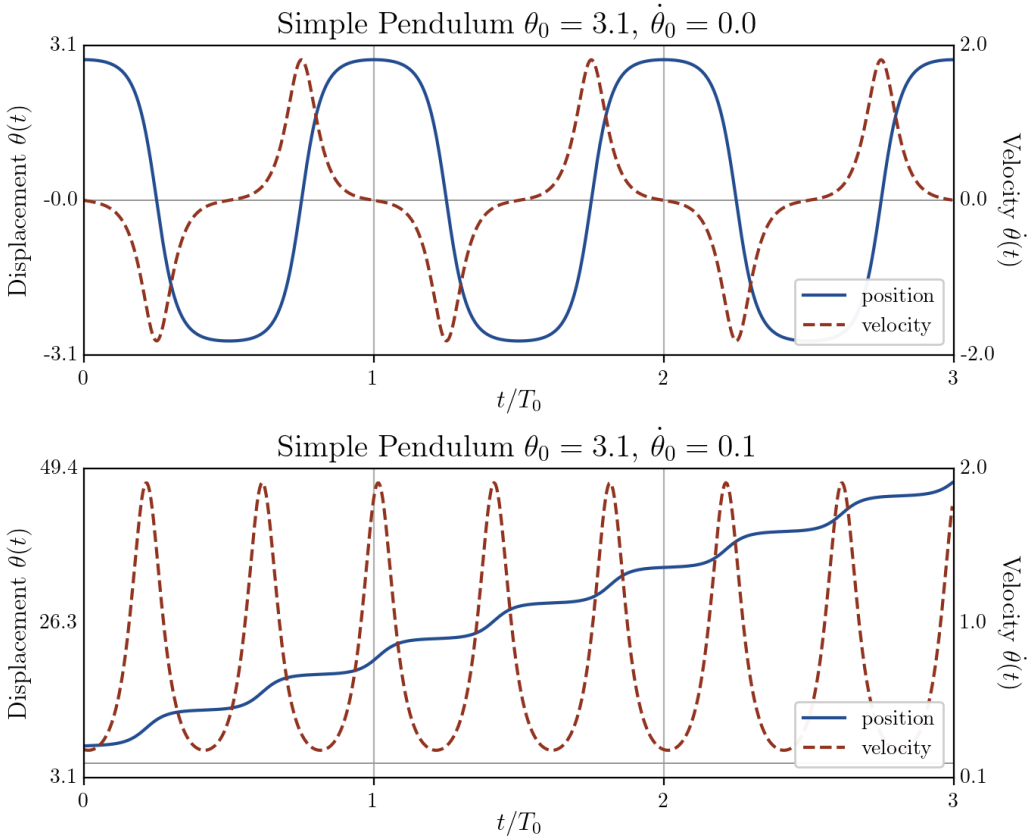

Figure 2: The simple pendulum's just-bound motion near the tip-over point with $(\theta_0, \dot{\theta}_0) = (0.99\pi, 0)$ and free motion with $(\theta_0, \dot{\theta}_0) = (0.99\pi, 0.1)$. Note that both angular displacement θ and angular velocity $\dot{\theta}$ share a single plot. Found with `pefrl`.

3 Accuracy

For this report, I chose to classify the method by the use of a fixed or adaptive time step. There are of course many other ways group the methods.

3.1 Fixed-Step Methods

Figures 3 and 4 show the error in the simple pendulum's displacement and energy, respectively, over the course of an approximately 15-period simulation for various fixed-step numerical methods using the initial condition $\theta_0 = 1.0$ and $\dot{\theta}_0 = 0$. Displacement error is measured with respect to the analytic solution given in Subsection 2.2, while energy error is the deviation from the initial energy in Equation 1.
Note that the symplectic methods (shown in purple with diamond markers) maintain a roughly constant energy, while the energy in other fixed-step solutions diverges from the initial value over time. The 4th-order symplectic `pefrl` method outperformed all other fixed-step methods at a given step size, which seems reasonable for an energy-conserving system like the simple pendulum. However, the 2nd-order symplectic `verlet` was reliably outperformed by by the non-symplectic 4th-order Runge-Kutta methods `rk4` and `rk4r`.

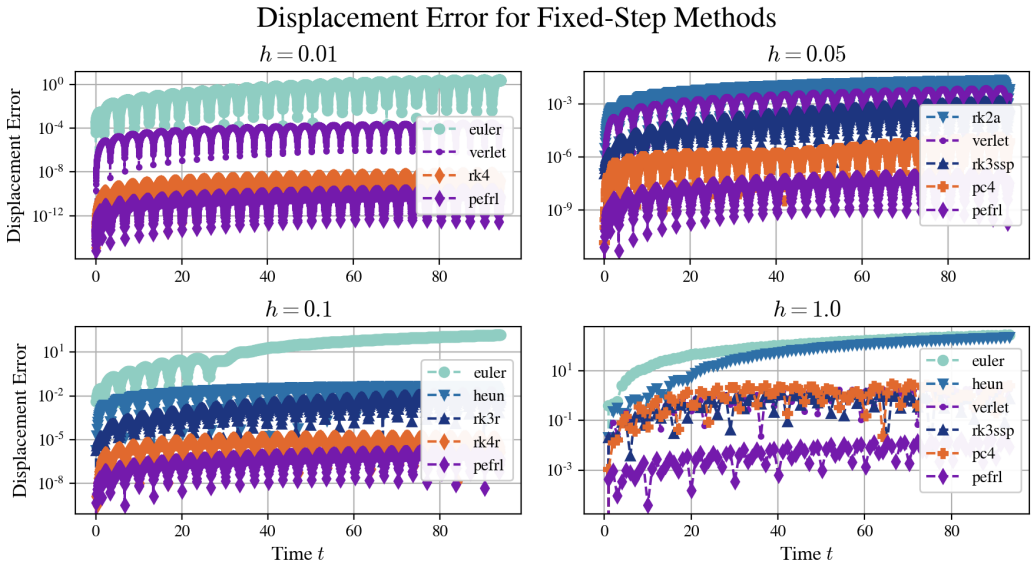

Figure 3: Error in angular displacement over time for various fixed-step numerical solutions to the simple pendulum's motion over approximately 15 periods for four step sizes h. Tested with $\theta_0 = 1.0$ and $\dot{\theta}_0 = 0$.

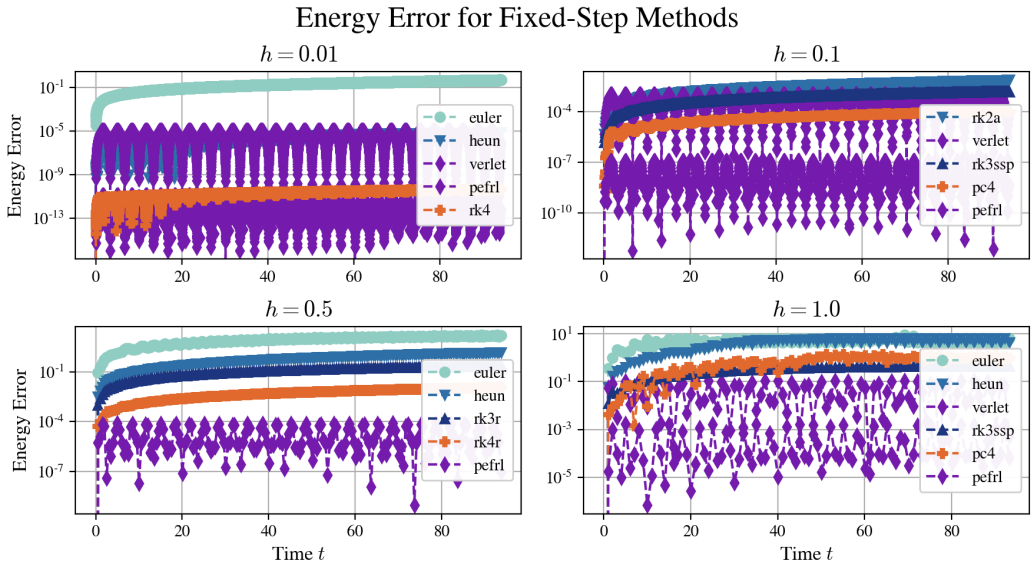

Figure 4: Error in energy (deviation from the initial energy) over time for various fixed-step numerical solutions to the simple pendulum's motion over approximately 15 periods for four step sizes h. The symplectic methods maintain a roughly constant error over time. Tested with $\theta_0 = 1.0$ and $\dot{\theta}_0 = 0$.

3.2 Adaptive-Step Methods

Figures 5 and 6 show the error in the simple pendulum's displacement and energy, respectively over the course of an approximately 15-period simulation for various

adaptive-step numerical methods using the initial condition $\theta_0 = 1.0$ and $\dot{\theta}_0 = 0$. The 4th order symplectic `pefrl`, with a step-size h approximately tailored to match adaptive-step tolerance ϵ, is shown for reference. As for the fixed step methods, displacement error is measured with respect to the analytic solution given in Subsection 2.2, while energy error is the deviation from the initial energy in Equation 1.
In general the built-in implicit methods `BDF`, `Radau` and `LSODA` from SciPy's `solve_ivp` perform relatively poorly at a given tolerance compared to explicit methods, but the poor accuracy should be taken with a grain of salt, since I did not specify a Jacobian matrix for use with the fixed-step methods and relied on the presumably less accurate built-in finite-difference approximation. That said, I was surprised to find `Radau`'s relative performance improved for large tolerances (see e.g. the plots with $\epsilon = 0.5$). Perhaps the chief takeaway is the symplectic `pefrl`'s method's excellent performance relative to the built-in methods from `solve_ivp`.

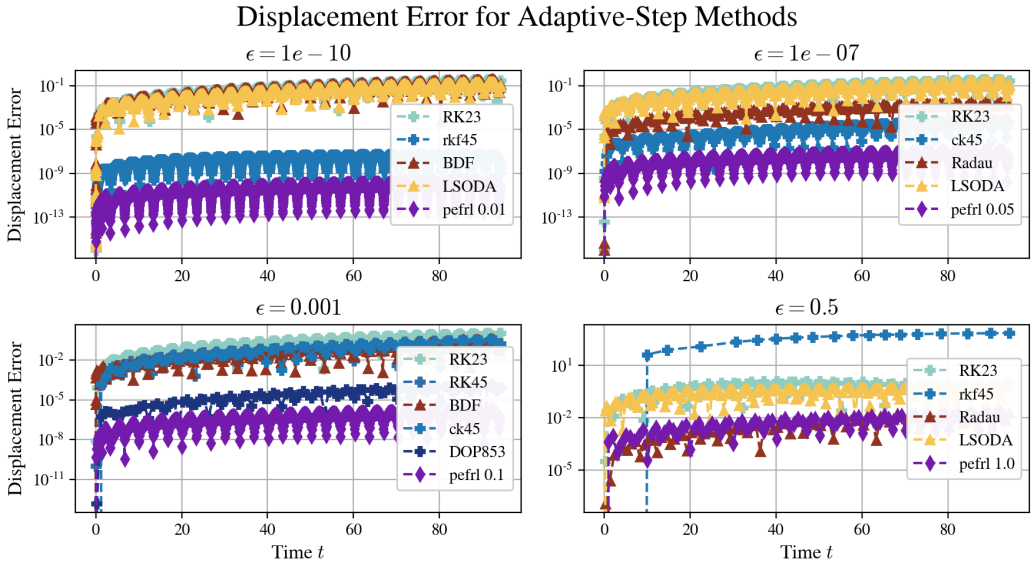

Figure 5: Error in angular displacement over time for adaptive-step solutions to the simple pendulum's motion over approximately 15 periods for four step sizes. Tested with $\theta_0 = 1.0$ and $\dot{\theta}_0 = 0$.

3.3 Brief Case Study: Built-In Methods Aren't Always Better

To test the energy conservation of symplectic methods, I simulated the simple pendulum's motion over 100 simulation using the 4th-order symplectic `pefrl`; `rk4`, a standard implementation of the canonical 4th-order fixed-step Runge-Kutta method; and SciPy's `odeint`, which is a wrapper for the `lsoda` method from the FORTRAN library `odepack`. I used `pefrl` and `rk4` with a step size of 0.05 and—intentionally—left `odeint` with its default settings.
Figure 7 shows the results of the 100-period simulation. `pefrl` performs superbly, showing negligible deviation from the initial energy over the entire simulation. Meanwhile, I was surprised to find `rk4` outperformed `odeint`! I must stress that I used `odeint` with its default settings (I'm sure appropriately adjusting its many param-

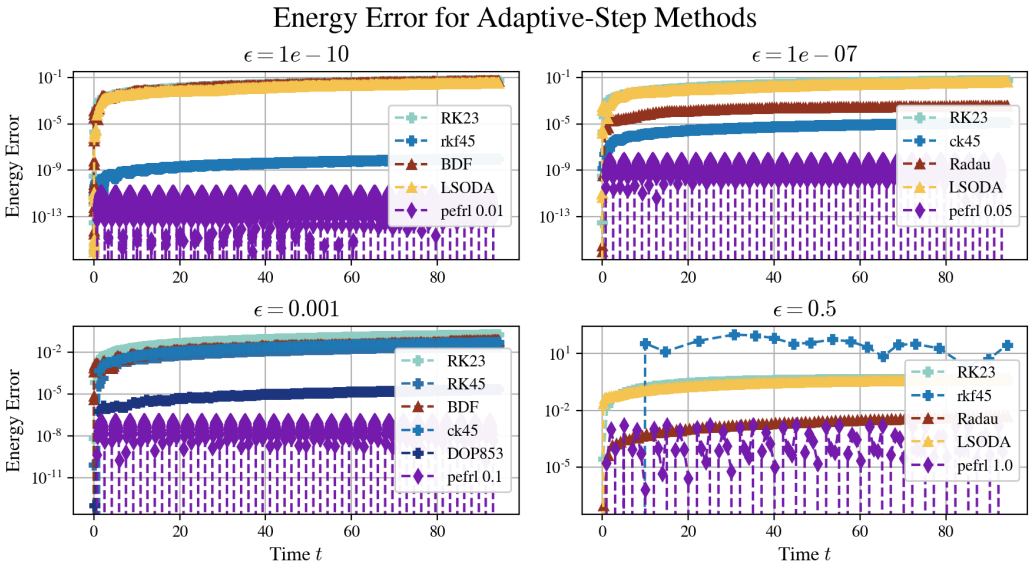

Figure 6: Error in energy (deviation from the initial energy) over time for various adaptive-step solutions to the simple pendulum's motion over approximately 15 periods for four error tolerance ϵ. The symplectic `pefrl`, using four corresponding step sizes, is shown for reference. Tested with $\theta_0 = 1.0$ and $\dot{\theta}_0 = 0$.

eters would produce a better result), but this was intentional. Namely, I wanted to demonstrate that blindly using built-in methods like `odeint` without consideration to the system at hand will not always produce a better result than even relatively simple "hand-built" methods well-suited to the system, e.g. `pefrl`.

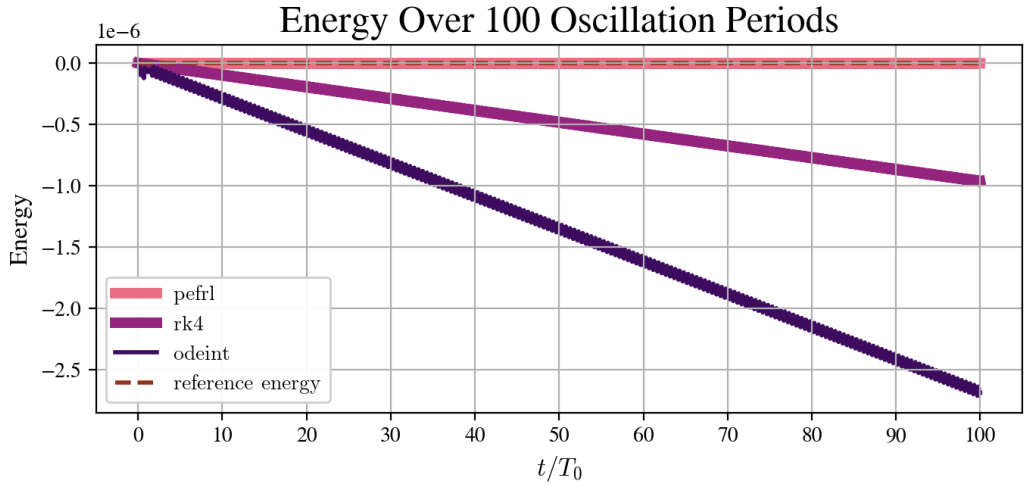

Figure 7: Energy deviation from the initial The symplectic `pefrl`, using four corresponding step sizes, is shown for reference. Tested with $\theta_0 = 1.0$ and $\dot{\theta}_0 = 0$.

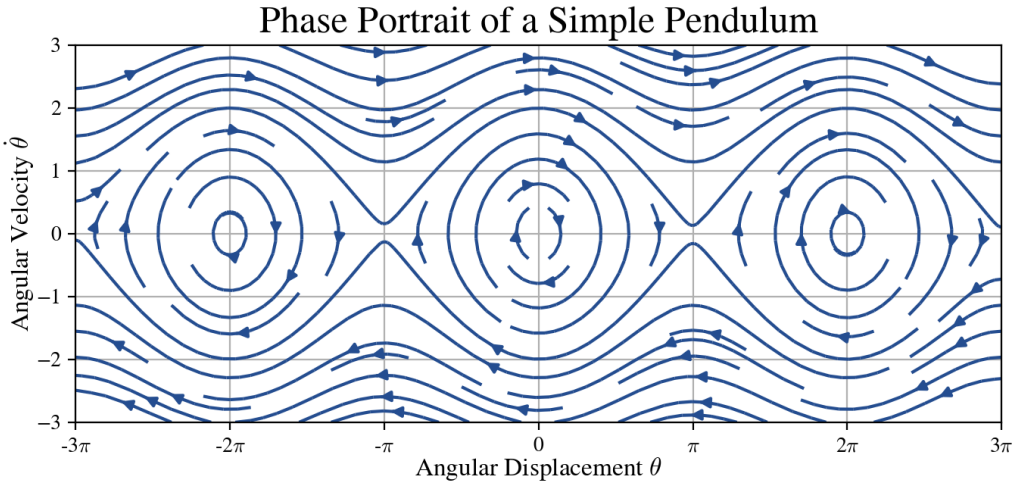

Figure 8: Phase portrait of a simple pendulum—note the distinction between bound and unbound states, which correspond to the circular and undulated streams.

4 Phase Portraits

All phase portraits are drawn with the same protocol: I defined angular displacement and angular velocity grids spanning $\theta \in [-3\pi, 3\pi]$ and $\dot{\theta} \in [-3, 3]$ on which to draw the portrait, and plotted the angular velocity and acceleration at each point on the grid using Matplotib's `streamplot` function. The angular and velocity and acceleration are found using the differential equation of state for each pendulum discussed in Subsection 2.3, which returns $\ddot{\theta}, \dot{\theta}$ at a given value of $\dot{\theta}, \theta$ and time. The following Python block shows the protocol for a generic function `get_pendulum_state`, which would be replaced with the function for the appropriate system.

```
from matplotlib import pyplot as plt
def plot_phase_space_example():
    x1D = np.linspace(-3*np.pi, 3*np.pi, 100)  # domain of displacements
    v1D = np.linspace(-3, 3, 100)  # domain of velocities
    xgrid, vgrid = np.meshgrid(x1D, v1D)  # 2D grids for x and v
    Vgrid, Agrid = np.zeros(np.shape(xgrid)), np.zeros(np.shape(vgrid))
    for i in range(np.shape(xgrid)[0]):
        for j in range(np.shape(xgrid)[1]):
            x = xgrid[i, j]
            v = vgrid[i, j]
            state = get_pendulum_state([x, v], 0)  # returns [v, a] at t=0
            Vgrid[i, j] = state[0]
            Agrid[i, j] = state[1]
    plt.streamplot(xgrid, vgrid, Vgrid, Agrid)  # draw with streamplot
    plt.show()
```

Figures 8 and 9 show the phase portraits of a frictionless and damped ($\beta = 0.5$) simple pendulum. Appendix A in shows phase portraits for both an un-driven and driven Van der Pol oscillator.

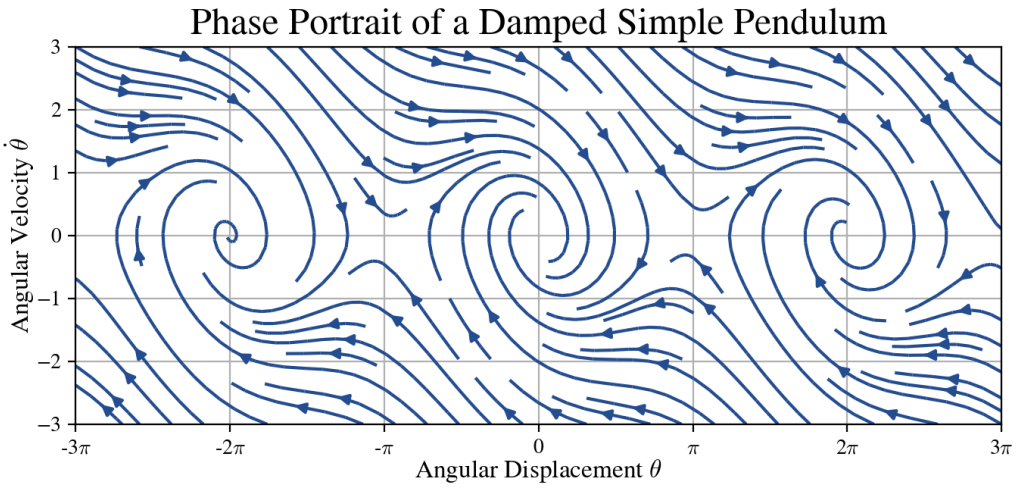

Figure 9: Phase portrait of a damped simple pendulum—note the spirals towards $\dot{\theta} = 0$, which correspond to damped energy loss. Tested with $\beta = 0.5$.

5 Damped, Driven Simple Pendulum

Figure 10 shows the phase space of a damped, driven simple pendulum, tested with $\beta = 0.5, \omega_d = 2/3$ and $A_d = 1.0$. Figure 11 shows the pendulum's resonance curve for various driving amplitudes A_d, which plot the pendulum's maximum amplitude over 40 oscillation periods as a function of driving frequency ω_d. I was frankly baffled by the resonance curve's behavior for driving amplitudes above $A_d = 1.0$, which show periodic divergence to pendulum amplitudes upwards of 400 and a plateau for $A_d = 1.5$ in the range $\omega_d \in (0.65, 0.8)$. The pendulum's resonance behavior ideally deserves further study, but, under time pressure and an abundance of other interesting tasks to preoccupy myself with, I did not investigate further.

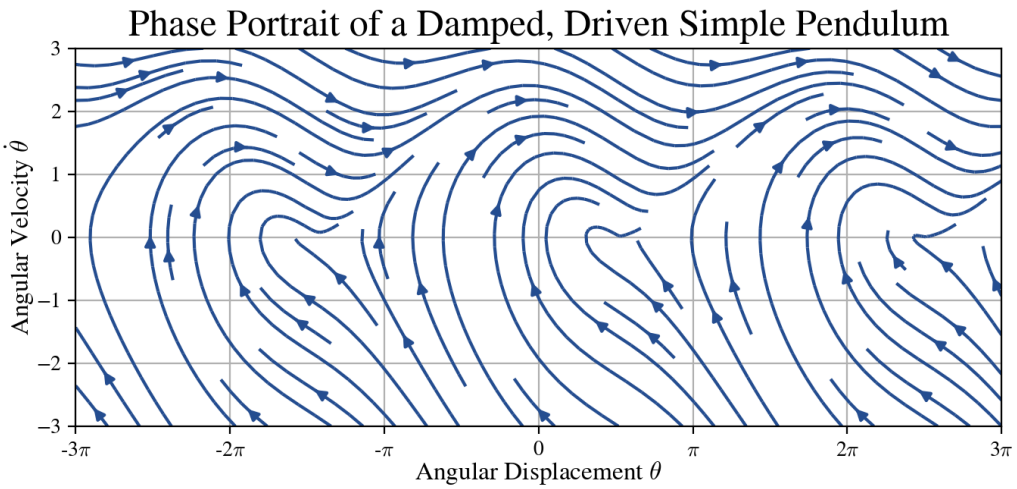

Figure 10: Phase portrait of a damped, driven simple pendulum. Tested with $\beta = 0.5, \omega_d = 2/3$ and $A_d = 1.0$.

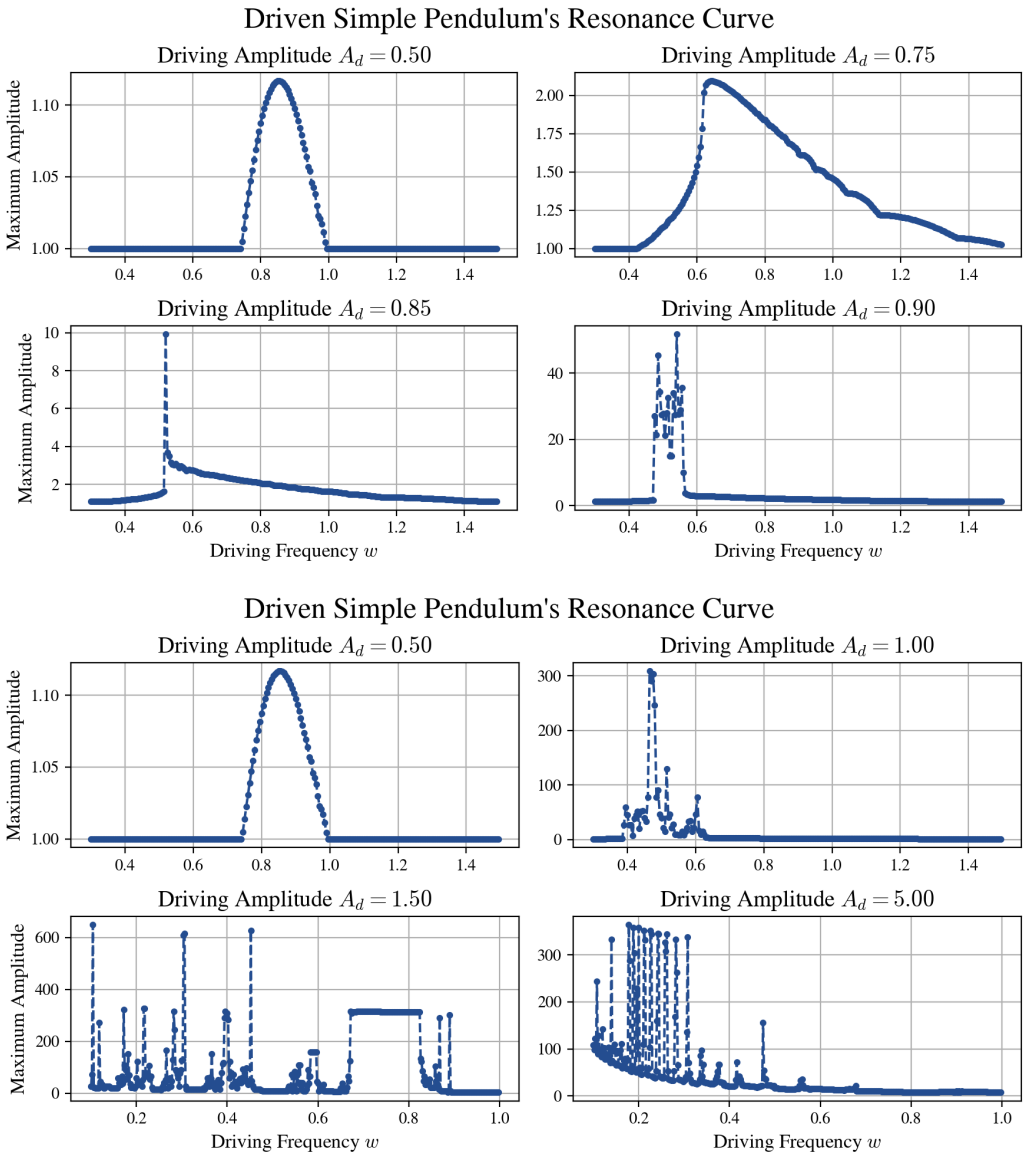

Figure 11: Resonance curve of a damped, driven simple pendulum for both small and large driving amplitudes. The plots show the pendulum's maximum amplitude over 40 periods as a function of driving frequency. Note the divergence for driving amplitudes above $A_d = 1.0$.

A Phase Portraits of the Van der Pol Oscillator

A.1 "Un-Driven" Van der Pol Oscillator

Figure 12 shows phase portraits of an un-driven Van der Pol oscillator for two values of the damping parameter λ. Both phase portraits decay toward the origin at $\theta, \dot{\theta} = (0, 0)$ which corresponds to damped energy loss. The decay for small damping with $\lambda = 0.1$ is gradual, while the decay is quite sharp for $\lambda = 1.0$.

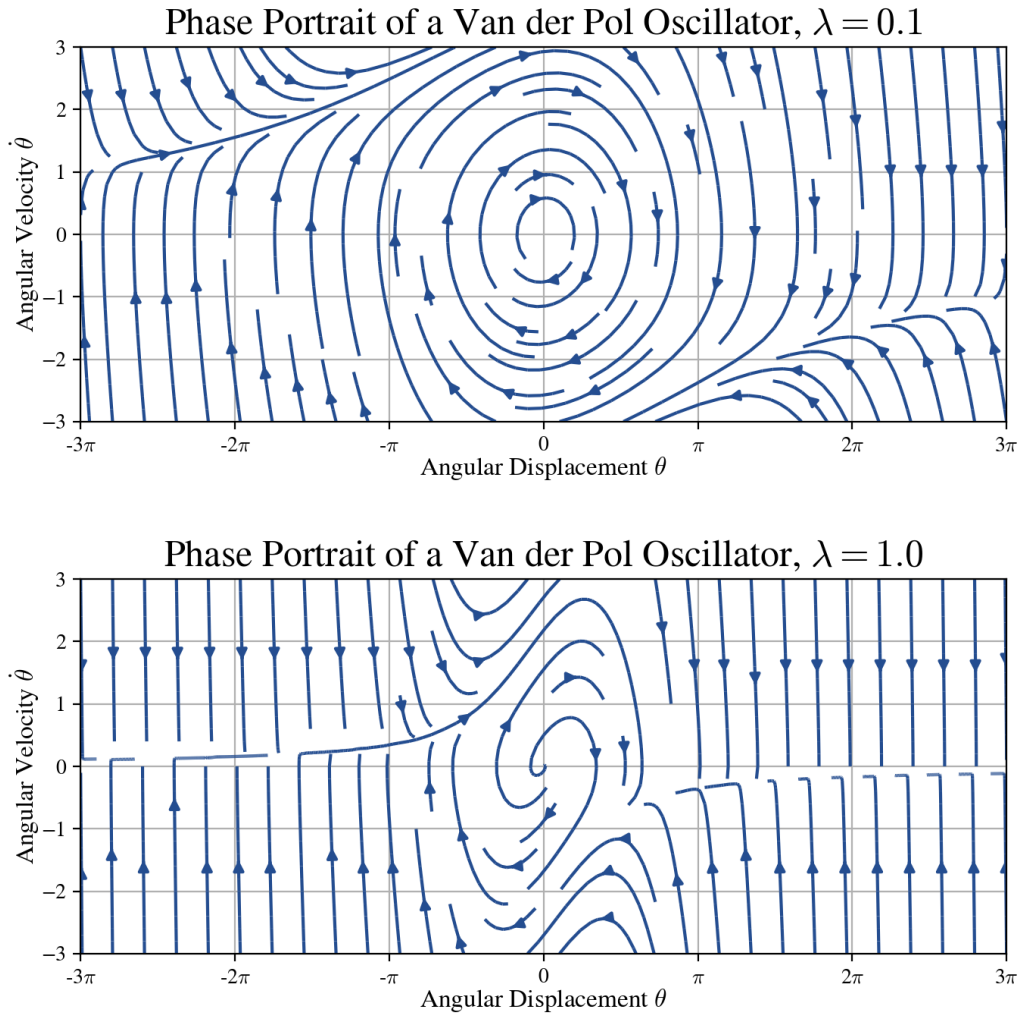

Figure 12: Phase portrait of an un-driven Van der Pol oscillator for two values of the damping parameter λ.

A.2 Driven Van der Pol Oscillator

Figure 13 shows phase portraits of a driven Van der Pol oscillator for three driving amplitudes A_d—the oscillator appears to diverge at large driving amplitudes.

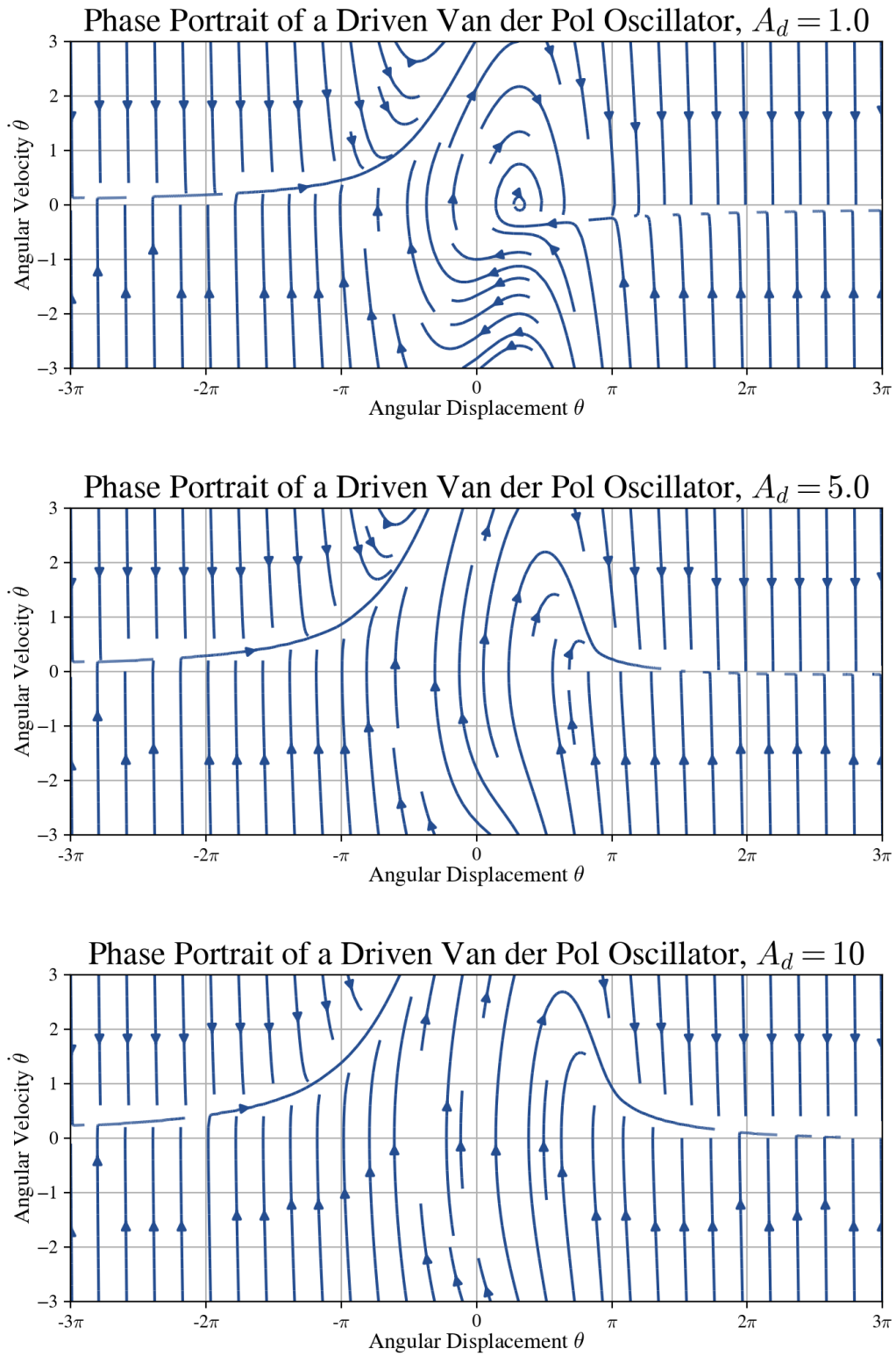

Figure 13: Phase portrait of a Van der Pol oscillator for three driving amplitudes A_d. Tested $\lambda = 1.0$ and $\omega_d = 1.0$.

B Extra: Simulating the Earth-Moon-Sun System

As an extra, in the spirit of Newton's second law, I tried simulating the Earth-Moon-Sun three-body system using the differential equation methods in this report. As a simplification, I assumed the sun was fixed in inertial space, and solved only for the motion of the earth and moon around the sun, which reduces the problem to a system of 12 equations (2 bodies $\times$ 3 spatial dimensions $\times$ 2nd order differential equation). The corresponding code is included in the file `newton-ems.py`. Figure 14 shows the earth and moon's orbit around the sun over the course of one year with Figure 15 shows the moon's 18.6 year nodal precession period. The attached files `ems-animated-365.mp4` and `moon-precession.mp4` show animated versions of the same graphs.

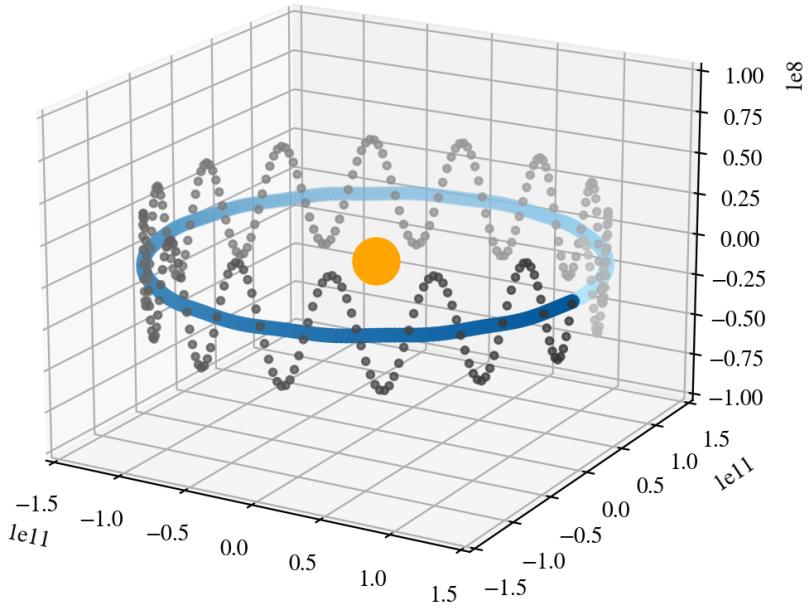

Figure 14: Motion of the earth and moon around the sun over the course of one year. The z axis is scaled down by three orders of magnitude to show the moon's z variation as it orbits the earth.

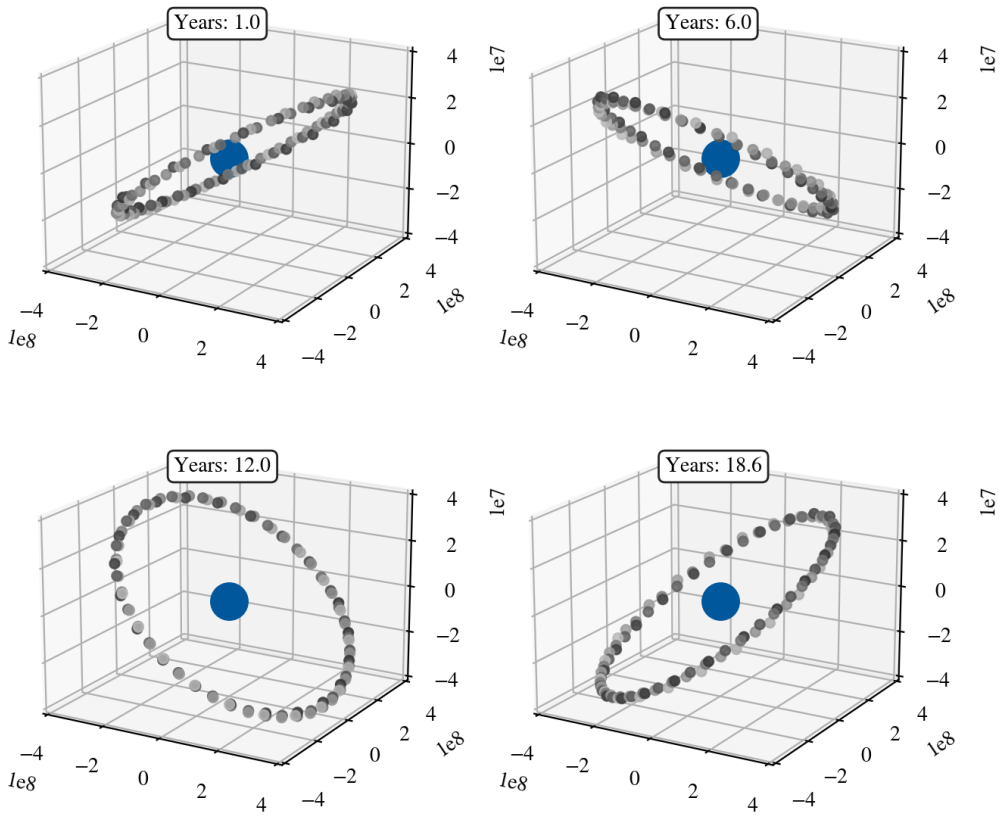

Figure 15: Visualizing the moon's 18.6 year nodal precession as it orbits the earth.

References

[1] Beléndez, Augusto, et al. "Exact solution for the nonlinear pendulum". Revista Brasileira de Ensino de Física. **29**, 645 (2007).

[2] Ochs, Karlheinz. "A comprehensive analytical solution of the nonlinear pendulum". European Journal of Physics. **32**, 479 (2011).

[3] Körber, Christopher and Hammer, Inka and Wynen, Jan-Lukas and Heuer, Joseline and Müller, Christian and Hanhart, Christoph. (2018). A primer to numerical simulations: The perihelion motion of Mercury. Physics Education. 53. 10.1088/1361-6552/aac487.

[4] Dr. David R. Williams. Moon fact sheet. https://nssdc.gsfc.nasa.gov/planetary/factsheet/moonfact.html.