Pendulum Experiment

David Kanter Eivin

8th April 2022

1 Introduction

Pendulums have for hundreds of years interested physics and industrialists alike. Not only are pendulums mystical to observe, they are immensely useful for mechanical timekeeping. From a physics standpoint, pendulums demonstrate the constant transformation between potential and kinetic energy, and, in a vacuum (with frictionless a frictionless pivot), can also demonstrate Newton's law of inertia by continuing to swing forever.

Pendulums rely on the force of gravity pulling down on a massive bob, counteracted by tensive forces on the massless string/rod. The result is angular acceleration towards the point of maximum kinetic energy. An oscillation period is the time it takes for the bob to move across the pendulum (to the opposite point of maximum potential energy) and return to the highest point where it started.

This project simulates a pendulum and uses the results of the simulation to discuss the parameters determining a pendulum's period. The nature of a simulation means that while results are very precise and reproducible, their accuracy is largely dependent on previous observations: in this case, the laws of angular motion and air resistance. The code is included in this report is written in the simplest terms possible; while a software background may be helpful, a physics background should be sufficient to *understand* the report *and* the simulation's source code.

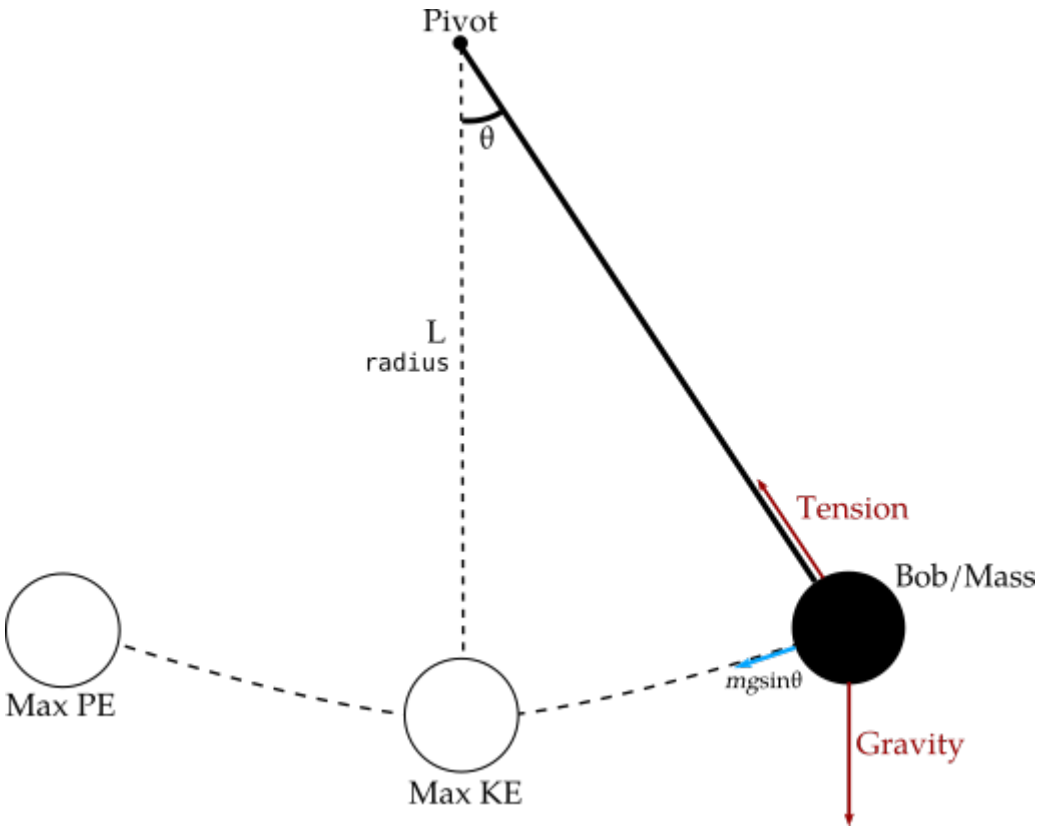


Figure 1: Pendulum

2 Purpose

Given the importance of the oscillation period, this experiment examines the dependence of the period on the pendulum's **maximum/starting amplitude** and **length**. Additionally, the experiment aims to **precisely simulate pendulum movement** with and without air resistance for reuse in other pendulum experiments.

3 Method

3.1 Software Simulation (Apparatus)

The Julia Programming Language was selected for its readable mathematical syntax for non-technical readers and excellent performance. To avoid the tedious and unclear computation of kinetic vectors using x and y coordinates, I used Lagrangian mechanics to work directly with angular quantities. After deriving formulas for instantaneous angular acceleration (α) and air resistance (shown below), I used Euler's method to compute the next angular velocity (ω) and position (θ). Essentially, angular acceleration is assumed to be constant for a small time interval Δt in order to compute ω. This is assumed to be constant for the same time interval, thus allowing the computation of θ at $t + \Delta t$. Time is then advanced by Δt and θ is once again used to compute α. **For all trials,** $\Delta t = 0.0001$s. Each step is recorded into a `DataFrame` to be available for analysis and animation.

3.1.1 Lagrangian

The simulation formulas start from a general differential equation for motion.

$$\frac{d}{dt}\frac{dL}{d\theta} = \frac{2L}{2\theta} - \frac{2D}{2\theta}$$

where L is the Lagrangian and D is the dampening factor. Then, where l represents the length of the rod:

$$\begin{aligned} L &= K - U \\ K &= \text{Kinetic Energy} = \frac{1}{2}mv^2 \\ &= \frac{1}{2}ml^2\theta^2 \end{aligned}$$

$$U = \text{Potential Energy} = mgh = mg(l - l \cdot \cos\theta$$

Together, this comprises the final Lagrangian:

$$L = \frac{1}{2}ml^2\theta^2 - mg(l - l \cdot \cos\theta)$$

which is differentiated with respect to θ to achieve:

$$\begin{aligned} \frac{d}{dt}\left(ml^2\dot{\theta}\right) &= -L\sin\theta mg \\ \implies \ddot{\theta} = \alpha &= -\frac{g\sin\theta}{l} \end{aligned}$$

As code:

```
ddθ(θ::Number) = -(sin(θ) * g) / radius
```

3.1.2 Air Resistance (drag)

Using the above formula, this simulation will continue forever. While this is useful for studying theory, it will differ substantially from experimental results, which are impacted substantially by air resistance.

To simulate air resistance, first I determined whether the linear or quadratic formulas would be more appropriate.

$$\begin{aligned} \Omega_{\text{linear}} &= bv = \beta Dv \\ \Omega_{\text{quadratic}} &= cv^2 = \gamma D^2 v^2 \end{aligned}$$

where D is the diameter of the bob. The bob is assumed to be a sphere with $\gamma = 0.25$. I also chose to measure velocity from the point of maximum PE.

$$
\begin{aligned}
mgh &= \frac{1}{2}mv^2 \\
\implies gl &= v^2 \\
\implies v &= \sqrt{gl}
\end{aligned}
$$

Then:

$$
\frac{\Omega_{\text{linear}}}{\Omega_{\text{quadratic}}} = \frac{\beta D\sqrt{gl}}{\gamma D^2\sqrt{gl}^2} = \frac{\beta}{\gamma D\sqrt{gl}}
$$

Using example values $D = 0.1\text{m}$ and $l = 0.1\text{m}$:

```
(1.6e-4) \ (γ * D * sqrt(g * l))
```

```
154.75850643017978
```

we have that the quadratic formula is substantially larger (\ is inverse division) so **we will use the quadratic formula for air resistance**.

After differentiating and simplifying as we did previously, the formula for α which accounts for drag is:

$$
\alpha = -\frac{g\sin\theta}{l} - \gamma D^2 l\omega^2
$$

Notably, this is *not* a vector but a magnitude, and so direction must be adjusted to oppose ω.

```
drag(ω::Number) = γ * (radius) * (diameter ^ 2) * (ω ^ 2) / mass
```

3.1.3 Final Code:

Using the above formulas, the iterative process for simulating data is summarized as follows (see Appendix A for full code).

```
# Initial θ and ω are required.
# θ is provided as a parameter, ω is assumed to be 0
ω = 0
for t in range(0, duration; step=Δt) # increments Δt as needed
    α = ddθ(θ) - copysign(drag(ω), ω)
    push!(df, [t, θ, ω, α]) # record state at this point
    ω += α * Δt
    θ += ω * Δt
end
```

3.2 Data Collection

To collect the relevant data, I first identified & filtered for only the extremes of θ using ω.

```
function extremes(df::AbstractDataFrame)
    extremes = DataFrame(first(df))
    for i in 3:nrow(df)
        if sign(df[i-1, :ω]) ≠ sign(df[i, :ω])
```

```
            push!(extremes, df[i-1, :])
        end
    end
    extremes
end
```

The first period is then taken as the difference between the 1st and 3rd extreme.

The simulation is run several times with different parameters to simulate the independent variables under study (amplitude, length/radius), and the period is recorded at each step. Average period is taken as the mean of the first 15 periods (arbitrarily chosen).

In order to run the simulation with and without air resistance, I simply set $\gamma = 0$ which removes this term from the equation for α.

4 Results

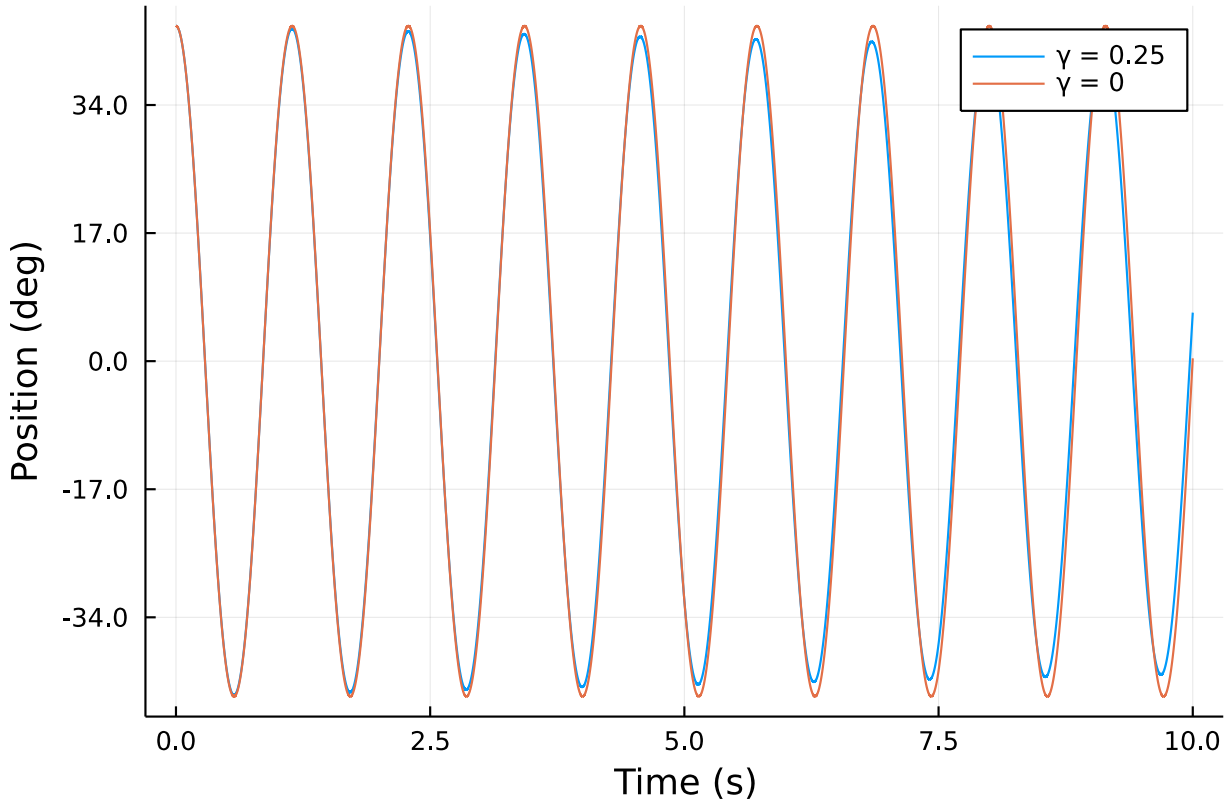
4.1 General/Simulation Effectiveness

Firstly, to validate our simulation for future tests, we can visualize the trajectory of our pendulum with and without air resistance.

```
p = Pendulum(
    .3, # 30cm pendulum
    deg2rad(45), # initial θ
    Bob(
        mass = .05, # 50g
        diameter = .05, # 5cm
        γ = γ
    )
)

with_resistance = simulate(p, 10)
no_resistance = simulate(p, 10, resistance=false)

plot(
    with_resistance[!, :time],
    [with_resistance[!, :θ], no_resistance[!, :θ]],
    label = ["\\gamma = $γ" "\\gamma = 0"],
    xlabel = "Time (s)",
    ylabel = "Position (deg)",
    yformatter = round ∘ rad2deg
)
```



We can see that the pendulum in a vacuum has constant amplitude, whereas the simulation with air resistance displays a slight dampening effect. This effect is likely so pronounced because the bob is light (50.0g).

4.2 Amplitude and Period

As we can see from the below figure, as amplitude increases, the time of the first period (roughly equivalent to the period in a vacuum) increases as well. This effect is very minor below 30° but becomes very pronounced above ~80 degrees. The ability to simulate such high angles with "rod-like" behaviour (that is, with no flexibility in the string), is one advantage of using a computer simulation over a physical pendulum. While the average period also increases with amplitude, this effect is far less pronounced given that an increased velocity increases the dampening factor of air resistance. This curve is closer to what we would likely see in a real-life experiment on Earth.

```
df = DataFrame(amplitude=Number[], period_initial=Number[], period_average=Number[])

for deg in range(5, 175; step = 2)
    local p = Pendulum(.3, deg2rad(deg), Bob(.05, .05, γ))
    runthrough = simulate(p, 30) |> extremes |> periods
    @inbounds push!(df, [deg runthrough[1] mean(runthrough[1:min(20,
length(runthrough))])])
end

plot(
    df[!, :amplitude],
    [df[!, :period_initial] df[!, :period_average]],
    label = ["First Period" "Average Period"],
    xlabel = "Amplitude (deg)",
    ylabel = "Period (s)"
```

)

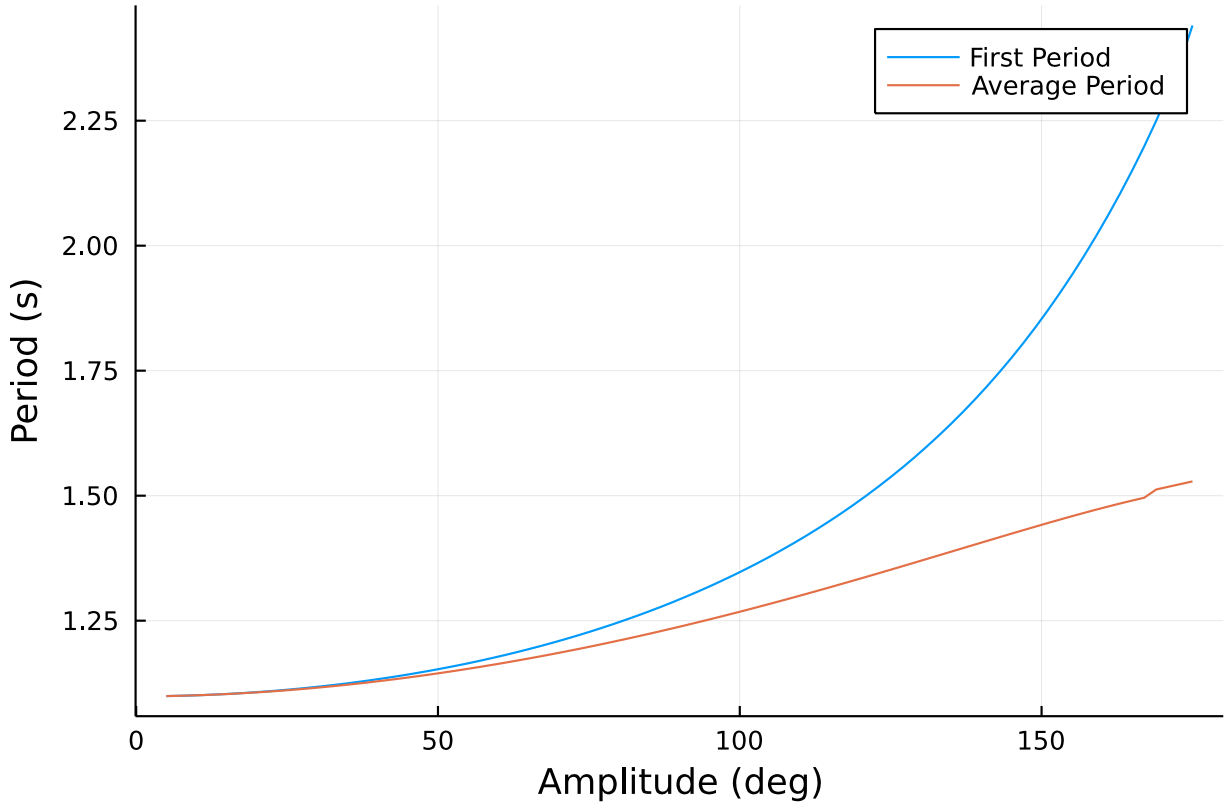


Figure 2: Oscillation period increases with release amplitude.

Showing the every 2 degrees of the first ~20 extremes gives a close up of the effect on small angles.

	amplitude	period_initial	period_average
	Number	Number	Number
1	5.0	1.0992	1.09928
2	7.0	1.0997	1.09977
3	9.0	1.1004	1.10041
4	11.0	1.1012	1.10121
5	13.0	1.1023	1.10215
6	15.0	1.1034	1.10326
7	17.0	1.1048	1.1045
8	19.0	1.1063	1.10589
9	21.0	1.108	1.10742
10	23.0	1.1098	1.10909
11	25.0	1.1119	1.1109
12	27.0	1.1141	1.11285
13	29.0	1.1165	1.11494
14	31.0	1.1191	1.11715
15	33.0	1.1218	1.1195
16	35.0	1.1247	1.12199
17	37.0	1.1279	1.1246
18	39.0	1.1312	1.12735
19	41.0	1.1347	1.13021
20	43.0	1.1384	1.13322

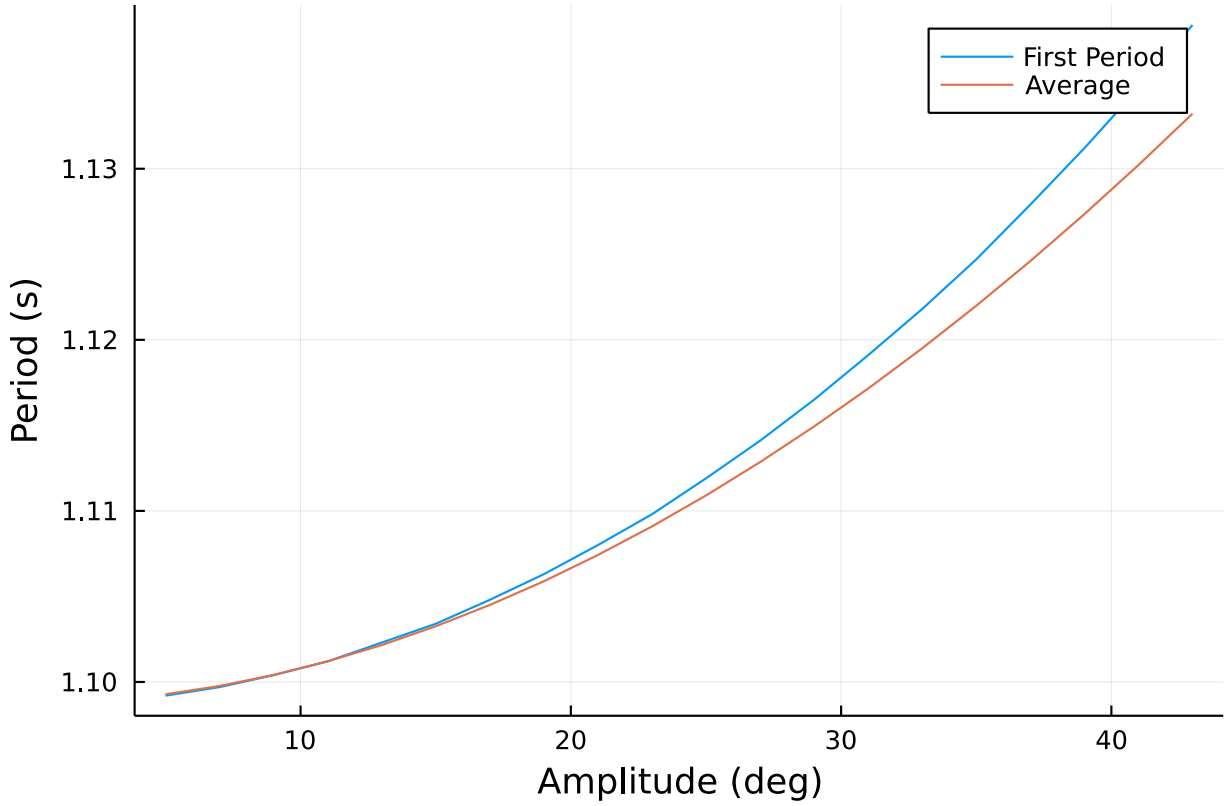


Figure 3: Period of First 20 Extremes: Detailed Image of Small Max Amplitudes.

This supports Galileo's hypothesis that amplitude has minimal/negligible impact on period at very small angles, but as angle increases, it starts to show very relevant effects.

4.3 Length (radius) and Period

Period also increase substantially as length is increased, from 10cm to 2m, though the rate of increase slows as the pendulum lengthens.

```
df = DataFrame(len=Number[], period_initial=Number[], period_average=Number[])

for len in range(.1, 2, step=.1)
    local p = Pendulum(len, deg2rad(30), Bob(.05, .05, γ))
    runthrough = simulate(p, 30) |> extremes |> periods
    @inbounds push!(df, [len runthrough[1] mean(runthrough[1:min(20,
length(runthrough))])])
end

plot(
    df[!, :len],
    [df[!, :period_initial] df[!, :period_average]],
    label = ["First Period" "Average"],
    xlabel = "String Length (m)",
    ylabel = "Period (s)"
)
```

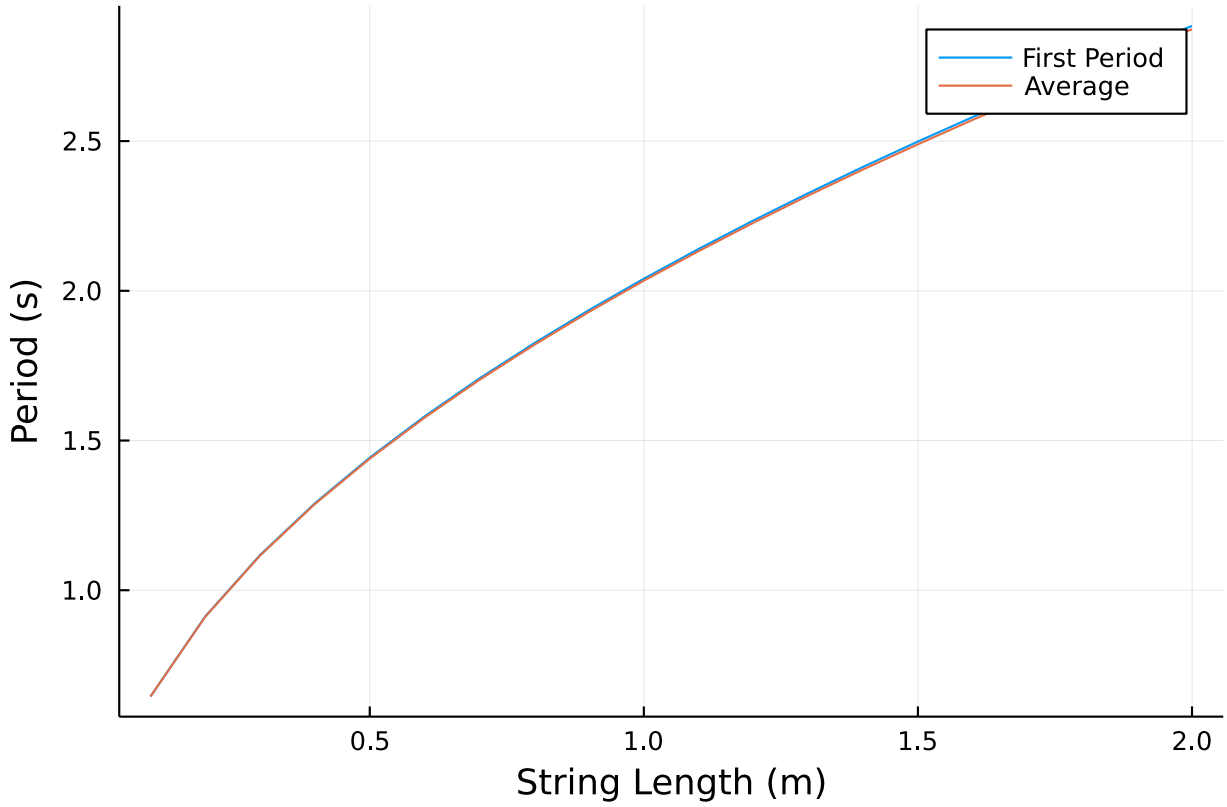


Figure 4: Period increases with penulum length (radius) at a release angle of 30deg.

To better visualize the effects of air resistance we can increase θ to increase the dampening effect. Once again, air resistance reduces the period as length increases, for the same reason as above.

This supports Galileo's hypothesis that length *does* affect period.

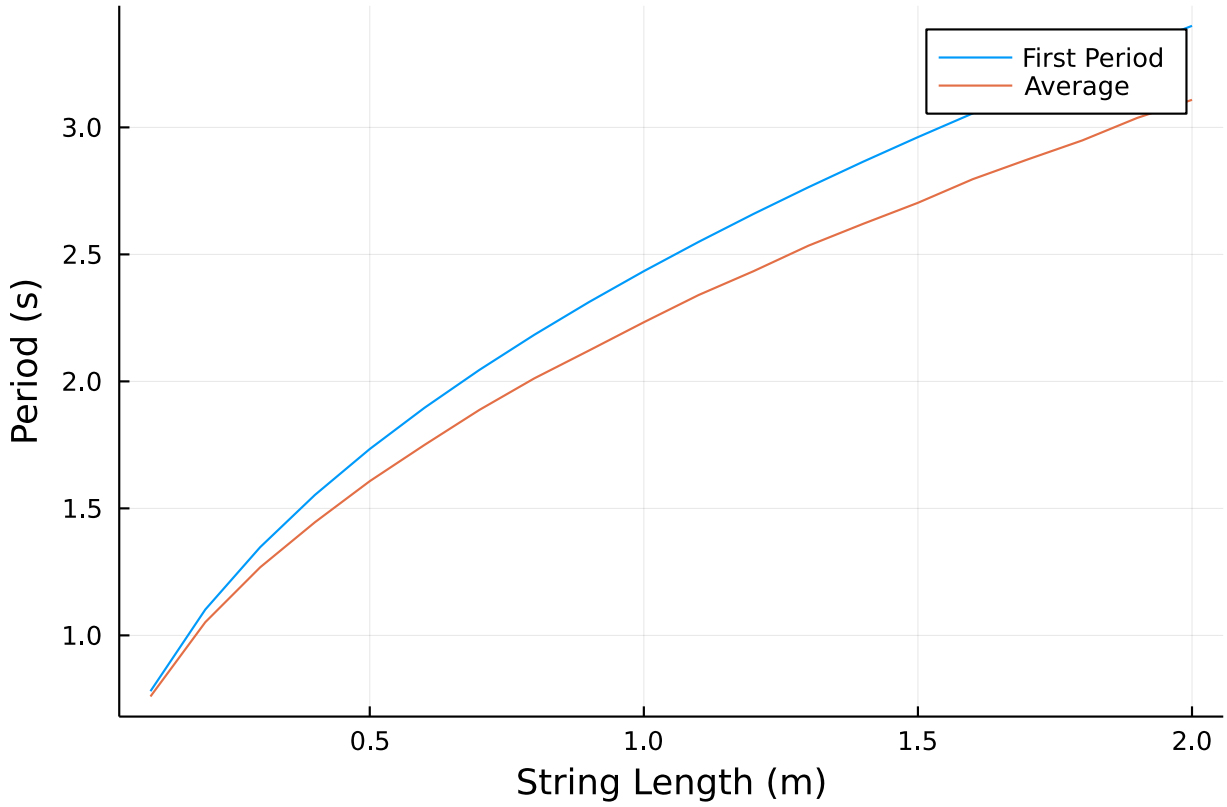


Figure 5: Period increases with penulum length (radius) at a release angle of 100deg.

5 Error, Uncertainty, and Limitations

Given the simulated nature of the experiment, and that it uses precise formulas without any random parameters, it is completely deterministic. Repeating the simulation will neccesarily yield the same results every time. Consequently, there is no uncertainty in the data or figures. This simulation is "perfectly" precise.

This simulation is very accurate as well; while there are very few sources of error (systemic and human), there exist some very minor sources of error inherent to the method.

Firstly, Euler's method is *not* precise integration, and as such the simulation "overshoots" the exactly correct value for θ each period. However, this error is effectively mitigated by reducing the Euler step size to 0.0001 seconds. Based on experimental simulations of this value of $\Delta\mathtt{t}$, $\epsilon = \pm 10^{-7}$ from the actual. Additionally, this error cancels itself since the same Euler inaccuracy occurs at both extremes of the sin function.

In addition, by virtue of being a computer simulation that uses floating point values, there exists some negligible (and unavoidable) floating point inaccuracy which is typically $< 10^{-}8$ at the most. This inaccuracy is unpredictable, but it does not significantly impact the results.

It should be noted that while computer simulations are very useful for precise and quick simulation of physics processes, they rely on prior real-world experiments and observations. For example, γ is experimentally determined for spheres. To verify the experiment, I also compared its behaviour to real-life pendulum behaviour, to ensure that it behaved similarly.

6 Appendix A

The full source code for this project is available on GitHub: https://github.com/dkantereivin/pendulum-sim.

Plots are generated using Julia's Plots library (doi: 10.5281).