Team Goal:
• build the equations of motion using Lagrange and least action L=T-V • solve for the motion for a slow rotation speed and a fast rotation speed • visualize the solution with plots and animations
Team Repository:
project_01-12/project_01.jl at main · UConn-Dynamics/project_01-12
Julia Assistant:
build a julia solution using Symbolics to help me take a partial derivative of this Lagrangian $L=T-V$ where $T=1/2mL^2\dot{\omega}^2$ $v=mgL(1-cos(\theta))$ and the langrangian is $\partial /\partial t(\partial L/\partial \dot{\omega} - \partial L/\partial \theta$ Then create an animation with three plots, the angle, the speed and the animated pendulum swinging. Use the @gif plots command

Begin Julia/Pluto Code

```
using Symbolics,  Latexify       # symbolic math, derivatives, simplification
using DifferentialEquations  # solving the pendulum ODE
using Plots          # plotting and making GIFs (@gif)
```

Given

Frame dimensions:

- h1 = 0.2m
- w1 = 0.1m

Pendulum length:

- L=0.15m

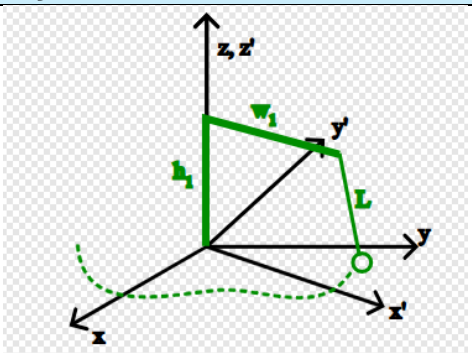
Pendulum mass:

- m=0.1kg (point mass at the end of the system).

Pendulum motion:

- swings in the xz-plane as it rotates at a constant speed omega

Geometry



Generalized position

- $\theta(t)$

Position of the Mass

$x = Lsin\theta$

$y = -Lcos\theta$

Velocity

$\omega_z = \dot{\theta}$

$\dot{x} = L\dot{\theta}cos\theta$

$\dot{y} = L\dot{\theta}sin\theta$

Kinetic Energy

$T = \frac{1}{2}mv^2$

- $v^2 = \dot{\bar{r}}^2$
- $\dot{\bar{r}}^2 = \dot{x}^2 + \dot{y}^2$
- $\dot{\bar{r}}^2 = (L\dot{\theta}cos\theta)^2 + (L\dot{\theta}sin\theta)^2$
- $\dot{\bar{r}}^2 = L^2\dot{\theta}^2(cos^2\theta) + L^2\dot{\theta}^2(sin^2\theta)$
- $\dot{\bar{r}}^2 = L^2\dot{\theta}^2(cos^2\theta + sin^2\theta)$
- $\dot{\bar{r}}^2 = L^2\dot{\theta}^2$

$T = \frac{1}{2}mL^2\dot{\theta}^2$

Potential Energy

$V = mgh$
$V = mg(\bar{r}\hat{j})$
$V = mg((-Lcos\theta) - (-L))$
$V = mg(-Lcos\theta + L)$
$V = mg(L - Lcos\theta)$
$V = mgL(1 - cos\theta)$

Lagrangian

$L = T - V$
$L = \left(\frac{1}{2}mL^2\dot{\theta}^2\right) - mgL(1 - cos\theta)$

Equation of Motion

$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\theta}}\right) - \frac{\partial L}{\partial \theta} = 0$
$\frac{\partial L}{\partial \dot{\theta}} = mL^2\dot{\theta}$
$\frac{d}{dt}\frac{\partial L}{\partial \dot{\theta}} = mL^2\ddot{\theta}$
$\frac{\partial L}{\partial \theta} = -mgLsin\theta$

Final Equation of Motion (EOM)

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\theta}}\right) - \frac{\partial L}{\partial \theta} = 0$$
$$(mL^2\ddot{\theta}) - (-mgLsin\theta) = 0$$
$$mL^2\ddot{\theta} + mgLsin\theta = 0$$

```
begin

    # --- Symbolic derivation of the pendulum Euler–Lagrange equation ---
    @variables time
    @parameters mass gravity length
    @variables angle(time)

    time_derivative = Differential(time)

    kinetic_energy = 0.5 * mass * length^2 * (time_derivative(angle)(time))^2
    potential_energy = mass * gravity * length * (1 - cos(angle(time)))
    lagrangian = kinetic_energy - potential_energy

    dL_dangle = expand_derivatives(differentiate(lagrangian, angle(time)))
    dL_dangledot = expand_derivatives(
      differentiate(lagrangian, time_derivative(angle)(time))
    )

    euler_lagrange = expand_derivatives(time_derivative(dL_dangledot) - dL_dangle)
    simplified_equation = simplify(euler_lagrange)

end
```

Plot

```
begin

    # --- Turn the symbolic result into a numeric ODE (nonlinear pendulum) ---
    pendulum_ode!(time_derivative_state, state, parameters, current_time) = begin
      angle_value = state[1]          # θ
      angular_velocity_value = state[2]  # θ̇

      gravity_value = parameters[:gravity]
      length_value = parameters[:length]

      time_derivative_state[1] = angular_velocity_value
```

```
    time_derivative_state[2] = -(gravity_value / length_value) * sin(angle_value)
  end

  pendulum_parameters = Dict(:gravity => 9.81, :length => 0.15)
  initial_state = [0.7, 0.0]          # 0.7 rad ≈ 40°, released from rest
  time_span = (0.0, 10.0)             # simulate for 10 seconds

end
```

```
begin

  # --- Solve the ODE using DifferentialEquations.jl ---
  pendulum_problem = ODEProblem(
    pendulum_ode!,
    initial_state,
    time_span,
    pendulum_parameters,
  )

  pendulum_solution = solve(
    pendulum_problem,
    Tsit5(),
    reltol = 1e-8,
    abstol = 1e-8,
  )

end
```

```
begin

  # --- Sample the solution to prepare data for plotting and animation ---
  time_samples = range(time_span[1], time_span[2], length = 300)

  angle_samples = [pendulum_solution(t)[1] for t in time_samples]
  angular_velocity_samples = [pendulum_solution(t)[2] for t in time_samples]

  length_value = pendulum_parameters[:length]
  pendulum_x = length_value .* sin.(angle_samples)
  pendulum_y = -length_value .* cos.(angle_samples)

end
```

```
begin

  # --- Create an animated GIF with angle, angular speed, and pendulum motion ---
  animation_file = @gif for index in eachindex(time_samples)
    angle_plot = plot(time_samples[1:index], angle_samples[1:index],
```

```
            xlabel = "time (s)", ylabel = "angle θ (rad)",
            title = "Pendulum Angle", legend = false)

        speed_plot = plot(time_samples[1:index], angular_velocity_samples[1:index],
            xlabel = "time (s)", ylabel = "angular speed θ̇ (rad/s)",
            title = "Angular Speed", legend = false)

        pendulum_plot = plot(xlims = (-1.2 * length_value, 1.2 * length_value),
            ylims = (-1.2 * length_value, 0.2 * length_value), ratio = :equal,
            xlabel = "x (m)", ylabel = "y (m)",
            title = "Pendulum Motion", legend = false)
        plot!(pendulum_plot, [0.0, pendulum_x[index]], [0.0, pendulum_y[index]], lw =
3)
        scatter!(pendulum_plot, [pendulum_x[index]], [pendulum_y[index]],
            markersize = 8, color = :red)

        plot(angle_plot, speed_plot, pendulum_plot, layout = (1, 3), size = (900, 300))
    end every 2
    ``


end
```

```
Saved animation to
C:\Users\medec\AppData\Local\Temp\jl_d4xWfGT8Ny.gif
```