From Lagrangians to Equations of Motion: A SymPy-Based Approach

Project 04

Lauren Sdun, Julia Jones, Julia Baumgarten

PHY 607 Computational Physics

December 9, 2025

- In classical mechanics we often derive equations of motion by hand.
- For simple systems this is fine, but:
  - algebra gets messy very quickly,
  - easy to make sign or factor mistakes,
  - hard to generalize to more degrees of freedom.
- Goal today:
  - Use **SymPy** to let the computer perform the Euler–Lagrange calculus.
  - Apply this to a 1D harmonic oscillator and a simple pendulum.

What is SymPy?

- Python library for **symbolic mathematics**.
- Can represent:
  - symbols (`m`, `k`, `g`),
  - functions of time (`x(t)`, `theta(t)`),
  - derivatives, integrals, algebraic expressions.
- Perfect for mechanics:
  - write a Lagrangian $L(q, \dot{q}, t)$,
  - let SymPy compute derivatives and simplify.

For a generalized coordinate $q(t)$ with Lagrangian $L(q, \dot{q}, t)$, the Euler–Lagrange equation is

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}}\right) - \frac{\partial L}{\partial q} = 0. \tag{1}$$

- Plan:
  1. Represent $q(t)$ and $\dot{q}(t)$ symbolically in SymPy.
  2. Build $L = T - V$.
  3. Compute the left-hand side of the Euler–Lagrange equation.
- SymPy then outputs the equation of motion automatically.

What `demo.py` Does

- Defines time and parameters:

$$t, \quad m, \quad k > 0$$

- Declares $x(t)$ as a symbolic function and computes $\dot{x}$.
- Builds the Lagrangian of the 1D harmonic oscillator:

$$L = \frac{1}{2}m\dot{x}^2 - \frac{1}{2}kx^2.$$

- Computes:
  - $\partial L/\partial x$,
  - $\partial L/\partial \dot{x}$,
  - $\frac{d}{dt}\big(\partial L/\partial \dot{x}\big)$,
  - $E_L = \frac{d}{dt}\big(\partial L/\partial \dot{x}\big) - \frac{\partial L}{\partial x}$.
- Simplifies E_L and prints

$$m\ddot{x} + kx = 0.$$

Key code from demo.py

```
t = sp.symbols('t')
m, k = sp.symbols('m k', positive=True)
x = sp.Function('x')(t)
xdot = sp.diff(x, t)

L = sp.Rational(1,2)*m*xdot**2 - sp.Rational(1,2)*k*x*

dL_dx     = sp.diff(L, x)
dL_dxdot = sp.diff(L, xdot)
d_dt_dL_dxdot = sp.diff(dL_dxdot, t)

EL = sp.simplify(d_dt_dL_dxdot - dL_dx)
print(EL)    # -> m*sp.diff(x, t, 2) + k*x
```

- This is the basic pattern we will reuse for the class problem.

Structure of Tutorial

- `pendulum_problem.py`:
  - contains TODOs to fill in
    (kinetic energy, potential energy, Lagrangian, etc).
- `pendulum_solution.py`:
  - Solution shown after the activity.

Physical Setup

- Point mass m at the end of a massless rod of length ℓ.
- Moves in a vertical plane under gravity g.
- Generalized coordinate: angle $\theta(t)$ measured from the downward vertical.
- We want the equation of motion using the Lagrangian method.

What You Will Implement

- Clone the Git repository and open `pendulum_problem.py`.
- Fill in the TODOs:
  1. define $\dot{\theta} = d\theta/dt$,
  2. write T and V,
  3. build $L = T - V$.
- Run

```
python pendulum_problem.py
```

  to print the symbolic equation of motion.
- We will walk around to help debug and interpret the output.

Useful SymPy Functions

- `sp.sin()` and `sp.cos()`
  - Symbolic versions of sin and cos.
  - Work with symbolic expressions such as `theta(t)`.

```
sp.sin(theta)
sp.cos(theta)
```

- `sp.subs()` (substitution)
  - Replaces one symbolic expression with another.
  - Used for the small-angle approximation $\sin\theta \approx \theta$.

```
expr.subs(old, new)
```

Example:

```
EL_small = EL_expr.subs(sp.sin(theta), theta)
```

- These functions let SymPy handle trigonometric expressions and symbolic approximations, which is essential for the pendulum problem.

Pendulum Lagrangian

- Coordinates and parameters:

$$\theta(t), \quad m, \quad \ell, \quad g.$$

- Kinetic energy:

$$T = \frac{1}{2} m\ell^2\dot{\theta}^2.$$

- Potential energy (zero at the bottom):

$$V = mg\ell(1 - \cos\theta).$$

- Lagrangian:

$$L = T - V.$$

Pendulum Equation of Motion

Using Euler-Lagrange equation we obtain:

$$m\ell^2\ddot{\theta} + mg\ell\sin\theta = 0$$

or, after dividing by $m\ell$,

$$\ell\ddot{\theta} + g\sin\theta = 0.$$

- Nonlinear equation: $\sin\theta$ term.
- **Small-angle approximation:** $\sin\theta \approx \theta$:

$$\ddot{\theta} + \frac{g}{\ell}\theta = 0,$$

which has the same form as the harmonic oscillator.

- **Define symbols:** time t, parameters m, ℓ, g, and the angle $\theta(t)$.
- **Compute energies:**
  - $T = \frac{1}{2} m\ell^2\dot{\theta}^2$
  - $V = mg\ell(1 - \cos\theta)$
- **Form the Lagrangian:** $L = T - V$
- **Apply Euler–Lagrange:** SymPy computes the symbolic EOM.
- **Simplify:** divide out $m\ell$ to obtain the standard form $\ell\ddot{\theta} + g\sin\theta = 0$.
- **Small-angle approximation:** substitute $\sin\theta \to \theta$ to get $\ddot{\theta} + \frac{g}{\ell}\theta = 0$.

Goal: Derive the equation of motion of a simple pendulum symbolically using SymPy and apply the small-angle approximation.

```
t = sp.symbols('t')
m, l, g = sp.symbols('m l g', positive=True)
theta = sp.Function('theta')(t)
theta_dot = sp.diff(theta, t)

T = sp.Rational(1,2) * m * l**2 * theta_dot**2
V = m * g * l * (1 - sp.cos(theta))
L = T - V

dL_dtheta = sp.diff(L, theta)
dL_dthetadot = sp.diff(L, theta_dot)
d_dt_dL_dthetadot = sp.diff(dL_dthetadot, t)

EL = sp.simplify(d_dt_dL_dthetadot - dL_dtheta)
EL_simpler = sp.simplify(EL / (m * l))
EL_small = sp.simplify(EL_simpler.subs(sp.sin(theta),
```

Why This is Useful

- SymPy automates the algebra in Lagrangian mechanics.
- Reduces human error and speeds up exploration of:
  - multi-degree-of-freedom systems,
  - constrained or coupled oscillators
- Today you:
  - saw how `demo.py` derives the SHO equation,
  - extended the method to a pendulum,
  - practiced writing and running symbolic code yourselves.

Thank you

Questions or comments?