LomaVerse - A Differentiable N-Body Gravitational Simulator in Loma(Checkpoint)

Samvrit Srinath
UC San Diego, USA
sasrinath@ucsd.edu

Eric Huang
UC San Diego, USA
eth003@ucsd.edu

Introduction – Recap (for progress update see section 3

N-body gravitational simulations are essential tools in astrophysics for modeling phenomena such as galactic evolution and planetary dynamics, as well as in aerospace engineering for designing spacecraft trajectories. For systems with more than two bodies ($n > 2$), closed-form solutions are generally unavailable, which motivates the use of numerical and gradient-based methods.

This project introduces `LomaVerse`[1], an N-body gravitational simulator developed using the Loma programming language. The primary objective is to leverage Loma's differentiability features, initially by simulating planetary motion, with the broader aim of enabling tasks like the optimization of interplanetary trajectories. The baseline contribution focuses on implementing a physics simulation for planetary motion by utilizing Loma's automatic differentiation capabilities.

Physical Model and Governing Equations

The dynamics of a system comprising n point masses interacting via gravity can be described using several formalisms. This project utilizes the Hamiltonian formulation, grounded in Newtonian Physics.

Hamiltonian Formulation

The project primarily explores the Hamiltonian approach. This formalism describes the system using generalized coordinates $\mathbf{q} = (\mathbf{r}_1, \ldots, \mathbf{r}_n)$ and their conjugate momenta $\mathbf{p} = (\mathbf{p}_1, \ldots, \mathbf{p}_n)$, where for Cartesian coordinates, $\mathbf{p}_i = m_i \mathbf{v}_i$ (with $\mathbf{v}_i$ being the velocity of body i). The system's dynamics are governed by the scalar Hamiltonian function $H(\mathbf{q}, \mathbf{p}, t)$, representing the total energy $H = T + U$.

The kinetic energy T is:

$$T(\mathbf{p}) = \sum_{i=1}^{n} \frac{||\mathbf{p}_i||^2}{2m_i}$$

The gravitational potential energy U is:

$$U(\mathbf{q}) = -\sum_{i=1}^{n} \sum_{j>i}^{n} G \frac{m_i m_j}{||\mathbf{r}_j - \mathbf{r}_i||}$$

Thus, the complete Hamiltonian for the N-body system is:

$$H(\mathbf{q}, \mathbf{p}) = \sum_{i=1}^{n} \frac{||\mathbf{p}_i||^2}{2m_i} - \sum_{i=1}^{n} \sum_{j>i}^{n} G \frac{m_i m_j}{||\mathbf{r}_j - \mathbf{r}_i||}$$

. Note that for numerical stability in implementation, the distance term $||\mathbf{r}_j - \mathbf{r}_i||$ is typically replaced by a softened version, such as $\sqrt{||\mathbf{r}_j - \mathbf{r}_i||^2 + \epsilon^2}$.

The time evolution of the system is described by Hamilton's equations:

$$\dot{\mathbf{r}}_i = \frac{\partial H}{\partial \mathbf{p}_i}$$
$$\dot{\mathbf{p}}_i = -\frac{\partial H}{\partial \mathbf{r}_i}$$

. These equations form the basis for the simulation implemented using Loma's AD capabilities.

1 Physical Model

The dynamics of a system of n point masses interacting gravitationally can be described through several equivalent formalisms. Here we describe 2 approaches: 1) the Newtonian Approach; 2) The Hamiltonian Approach.

1.1 Newtonian Formulation

In the Newtonian framework, interactions are described by forces. For any two bodies i and j with masses m_i and m_j and position vectors $\mathbf{r}_i$ and $\mathbf{r}_j$ respectively, the gravitational force exerted on body i by body j is given by its vector form:

$$\mathbf{F}_{ij} = G \frac{m_i m_j}{||\mathbf{r}_j - \mathbf{r}_i||^3} (\mathbf{r}_j - \mathbf{r}_i)$$

G is the gravitational constant. The magnitude of this force is $F_{ij} = G\frac{m_i m_j}{d^2}$, where $d = ||\mathbf{r}_j - \mathbf{r}_i||$ is the scalar distance.

The equation of motion for body i is obtained by summing the vector forces from all other bodies $j \neq i$: $m_i \ddot{\mathbf{r}}_i = \sum_{j \neq i}^{n} \mathbf{F}_{ij}$. This yields the acceleration of body i as:

$$\ddot{\mathbf{r}}_i = \sum_{j \neq i}^{n} G \frac{m_j}{||\mathbf{r}_j - \mathbf{r}_i||^3} (\mathbf{r}_j - \mathbf{r}_i)$$

These n coupled second-order ordinary differential equations (ODEs) are converted into a system of $2n$ first-order ODEs by defining the state of each body k as $(\mathbf{r}_k, \mathbf{v}_k)$, where $\mathbf{v}_k = \dot{\mathbf{r}}_k$ is its velocity. This formulation is implemented in

[1] Assuming `LomaVerse` is LomaVerse as per the document context.

LomaVerse by directly coding the force calculations to obtain accelerations, which are then numerically integrated. (V1)

1.2 Hamiltonian Formulation

We will also be exploring the usage of Hamiltonian mechanics. This formalism describes the system's state using generalized coordinates $\mathbf{q} = (\mathbf{r}_1, \ldots, \mathbf{r}_n)$ and their conjugate momenta $\mathbf{p} = (\mathbf{p}_1, \ldots, \mathbf{p}_n)$, where for Cartesian coordinates, $\mathbf{p}_i = m_i \mathbf{v}_i$. The system's dynamics are governed by a scalar function, the Hamiltonian $H(\mathbf{q}, \mathbf{p}, t)$, which in our usecase (gravitational N-body interactions), represents the total energy $H = T + U$.

The kinetic energy T in terms of momenta is:

$$T(\mathbf{p}) = \sum_{i=1}^{n} \frac{||\mathbf{p}_i||^2}{2m_i} \tag{1}$$

The gravitational potential energy U is:

$$U(\mathbf{q}) = -\sum_{i=1}^{n} \sum_{j>i}^{n} G \frac{m_i m_j}{||\mathbf{r}_j - \mathbf{r}_i||} \tag{2}$$

The Hamiltonian $H = T + U$ is thus:

$$H(\mathbf{q}, \mathbf{p}) = \sum_{i=1}^{n} \frac{||\mathbf{p}_i||^2}{2m_i} - \sum_{i=1}^{n} \sum_{j>i}^{n} G \frac{m_i m_j}{||\mathbf{r}_j - \mathbf{r}_i||} \tag{3}$$

The time evolution of the system is described by Hamilton's equations, a set of $2 \times (3n)$ first-order ODEs:

$$\dot{\mathbf{r}}_i = \frac{\partial H}{\partial \mathbf{p}_i} \quad \left(= \frac{\mathbf{p}_i}{m_i} = \mathbf{v}_i \right) \tag{4}$$

$$\dot{\mathbf{p}}_i = -\frac{\partial H}{\partial \mathbf{r}_i} \quad \left(= -\frac{\partial U}{\partial \mathbf{r}_i} = \mathbf{F}_i^{\text{grav}} \right) \tag{5}$$

2 Differentiable Simulation in Loma – Recap

The objective of LomaVerse is to implement a differentiable N-body simulator using the Loma programming language. While our initial implementation suits the direct Newtonian formulation for forces, Loma's AD capabilities also enable an elegant derivation of the equations of motion via the Hamiltonian formalism, similar to the approach demonstrated for a single pendulum in Homework 1's Pendulum Example.

2.1 Loma for N-Body Dynamics

Recalling the N-body Hamiltonian $H(\mathbf{q}, \mathbf{p})$ from Eq. 3:

$$H(\mathbf{q}, \mathbf{p}) = \sum_{i=1}^{n} \frac{||\mathbf{p}_i||^2}{2m_i} - \sum_{i=1}^{n} \sum_{j>i}^{n} G \frac{m_i m_j}{||\mathbf{r}_j - \mathbf{r}_i||_{soft}}$$

where $\mathbf{q} = (\mathbf{r}_1, \ldots, \mathbf{r}_n)$ and $\mathbf{p} = (\mathbf{p}_1, \ldots, \mathbf{p}_n)$. Hamilton's equations (Eqs. 4-5) dictate the system's evolution: $\dot{\mathbf{r}}_i = \partial H / \partial \mathbf{p}_i$ and $\dot{\mathbf{p}}_i = -\partial H / \partial \mathbf{r}_i$.

Loma's AD can be used to compute these partial derivatives. First, we would implement H as a Loma function:

To get $\partial H / \partial p_{k,\alpha}$ (the α-component of $\partial H / \partial \mathbf{p}_k$), we would call ‘d_n_body_hamiltonian‘ by setting the ‘dval‘ for $p_{k,\alpha}$ to 1.0 and all other ‘dval‘s (for other momenta components and all position components) to 0.0. The ‘dval‘ of the output of ‘d_n_body_hamiltonian‘ would then be $\partial H / \partial p_{k,\alpha}$. A similar process yields $-\partial H / \partial r_{k,\alpha}$.

2.2 Differentiating the Simulation Process

Beyond deriving ODE terms, a goal is to differentiate the *entire simulation process* (i.e., the numerical solution of the ODEs over time) to obtain sensitivities of the final trajectory with respect to initial configuration of the system.

- **Forward Mode AD on Simulation**: For a simulation $S(\text{inputs}) \rightarrow \text{outputs}$, forward AD computes how ‘outputs‘ change with respect to a perturbation in one ‘input‘. We aim to use this for sensitivity analysis (e.g., $\partial \mathbf{r}_{\text{final}} / \partial \mathbf{v}_{\text{initial},k}$).
- **Reverse Mode AD on Simulation**: For a scalar loss function $L(\text{outputs})$ (i.e. how far are we from our objective.) In Path Optimization, think for a probe or a spacecraft, either we can simulate where the probe would be, OR compute reverse_mode w.r.t to our objective (i.e. "How far from Mars?"), reverse AD efficiently computes $\nabla_{\text{inputs}} L$.

Successfully differentiating the N-body simulation loop, using Hamiltonian-derived ODEs, is the *central technical challenge and contribution of this project.*

The Hamiltonian $H = T + U$ (total energy: $T = \sum_i \frac{\|\mathbf{p}_i\|^2}{2m_i}$, $U = -\sum_{i<j} G \frac{m_i m_j}{\|\mathbf{r}_j - \mathbf{r}_i\|_{\text{soft}}}$) often forms the basis for deriving the equations of motion or for energy conservation checks. These calculations would be embedded within the simulation step.

The code below presents a high level function representing a simulation run. The primary components are data structures for celestial bodies and simulation parameters, a main timestepping loop to evolve the system, and a scalar loss function if optimization is intended(i.e. if we had an objective specified).

High Level Implementation

```
// --- Core Data Structures (Conceptual) ---
struct Vec3 { x: float; y: float; z: float; }
struct BodyState { pos: Vec3; p: Vec3; mass:
    float; /* ... */ }
struct SimConfig { G: float; dt: float;
    num_steps: int; /* ... */ }
```

```
// --- Differentiable Simulation Function ---
def run_simulation_and_compute_loss(
    initial_body_states: In[Array[BodyState]],
         // Differentiable inputs
    config: In[SimConfig],                //
         Fixed parameters
    target_info: In[TargetData]           //
         e.g., target position
) -> float {
    current_states: Array[BodyState] =
         initial_body_states;

    // Main simulation loop (AD-compatible via
         max_iter)
    for step in range(config.num_steps) {
        // 1. Calculate forces/accelerations
             for all bodies
        //   (e.g., using Hamiltonian
             derivatives dH/dr, or Newtonian
             F=ma)
        //   forces =
             compute_all_forces(current_states,
             config);

        // 2. Update Current State of System
             using equations of motion for one
             time step (dt)
        //   current_states =
             update_one_step(current_states,
             forces, config.dt);
    }

    // 3. Compute a scalar loss based on final
         states and target
    //   loss_value =
         calculate_objective_function(current_states,
         target_info);

    return loss_value; // This scalar is
         target for AD
}
```

‘

This high-level structure is the target for Loma's `fwd_diff` or `reverse_diff` tools. For instance, to derive Hamilton's equations ($\dot{\mathbf{r}}_i = \partial H/\partial \mathbf{p}_i$, $\dot{\mathbf{p}}_i = -\partial H/\partial \mathbf{r}_i$), `fwd_diff` would be applied to a Loma function computing H. For optimizing initial conditions to reach a target, `fwd_diff` would be applied to our simulation function. to get gradients of our loss w.r.t to the initial configuration of the simulator.

3 Implementation Progress - New

Our implementation effort is structured to demonstrate the simulation's outputs through a visualization that shows the positions of planetary bodies. This work is divided into distinct functional areas which, for future development, could be modularized further for enhanced reusability and testing.

3.0.1 Physical Environment and GUI. This is where the physical state of the system is stored. This includes physical attributes such as the position, velocity, and acceleration of planetary bodies and also includes the tracking of time. Note here that this component does not at all understand the physics of the system. Rather, that role is delegated to the actual physics engine, which this environment will compose.

Currently, our environment implementation is written in Python and the state of the system is visualized using a simple PyGame GUI.

3.0.2 Partial Derivative Calculation and Physics Engine (Loma Component). The core physics engine, responsible for determining the evolution of the system, is where our Loma-based implementation slots in. The Python environment described above (Section 3.0.1) interfaces with this engine at each time step. The state of the physical system (current positions $\mathbf{r}_k$ and momenta $\mathbf{p}_k$ for each body k) is passed from the Python host to a C library compiled from Loma code (specifically, a module named `planetary_motion.py` in our development).

This Loma component calculates the necessary partial derivatives based on the Hamiltonian formulation (Eq. 3), which includes a softening factor ϵ for numerical stability:

$$H(\mathbf{q},\mathbf{p}) = \sum_{i=1}^{N} \frac{||\mathbf{p}_i||^2}{2m_i} - \sum_{i=1}^{N}\sum_{j>i}^{N} G\frac{m_i m_j}{\sqrt{||\mathbf{r}_j - \mathbf{r}_i||^2 + \epsilon^2}}$$

Loma's ForwardDiffMutator class is applied to the Loma implementation of this Hamiltonian. This automatically generates a function `d_n_body_hamiltonian` capable of computing the Hamiltonian's value and its derivatives. Helper functions within the Loma module (e.g., `get_dH_dr_k_alpha`) then utilize `d_n_body_hamiltonian` to compute the specific partial derivatives $\partial H/\partial \mathbf{r}_k$ and $\partial H/\partial \mathbf{p}_k$ needed for Hamilton's equations (Eqs. 4-5). This approach bypasses the need for manual derivation of these potentially complex derivatives, a key merit of using Loma's AD features, especially as the number of bodies N increases. The compiled Loma module returns these derivatives to the Python host, which then applies them in the state update step.

Our implementation supports a configurable number of bodies N (up to a compiled maximum of `MAX_N_BODIES_CONST = 20`) (this limit is arbitrary), passed from the Python host via a `SimConfig` structure. The Loma physics calculations correctly iterate only over the active `config.num_bodies`.

3.0.3 State Updates. A naive approach to update the state of the system involves using Euler's method, which uses only one previous derivative to update the new state of a variable, such as velocity here: $\mathbf{v}_{\text{new}} = \mathbf{v}_{\text{old}} + \mathbf{a}\Delta t$. This approach is prone to numerical errors, especially if the simulation is left to run over a long period of time. By separating the logic for calculating and updating the derivatives of the system, we are able to include more complex logic for using the derivatives previously to update the system.

For our Hamiltonian system, we have implemented the **Symplectic Euler method** within the Loma physics engine (the `time_step_system` function). This first-order integrator is chosen for long-term stability and energy behavior in Hamiltonian dynamics. The update rules for positions $\mathbf{r}_k$ and momenta $\mathbf{p}_k$ over a time step Δt are:

$$\mathbf{p}_k(t + \Delta t) = \mathbf{p}_k(t) - \Delta t \cdot \frac{\partial H}{\partial \mathbf{r}_k}(\mathbf{r}(t), \mathbf{p}(t)) \tag{6}$$

$$\mathbf{r}_k(t + \Delta t) = \mathbf{r}_k(t) + \Delta t \cdot \frac{\partial H}{\partial \mathbf{p}_k}(\mathbf{r}(t), \mathbf{p}(t + \Delta t)) \tag{7}$$

The crucial aspect is the use of the newly computed momentum $\mathbf{p}_k(t+\Delta t)$ when evaluating the derivative for the position update, which makes the method symplectic. This method generally provides better energy conservation (bounded error) over long simulations compared to the naive Euler method. While a more complex scheme like 4th order Runge-Kutta (RK4) is yet to be explored, and has the potential to be more optimal.

3.1 Initial Findings and Demonstrations

Through the integration of the Loma-based physics engine with the Python host environment (visualized with `matplotlib` in our current iteration, adaptable to PyGame), we have successfully simulated various N-body configurations in 2D. Our findings confirm that the system can model:

- **Stable Solar System Analogs:** Simulations with 9 bodies (Sun + 8 planets) using appropriate masses and initial conditions. demonstrate relatively stable orbital patterns over extended periods (e.g., 50-75 simulated years). The long orbital trails visually depict these "hypothetical orbits" which are the result of the full N-body interactions. Over very long terms, subtle N-body perturbations (deviations from perfect Keplerian ellipses) become apparent in these trails, showcasing the mutual gravitational influences.
- **Smaller Body Systems:** We have also tested configurations with fewer bodies (e.g., 3-body and 4-body systems derived from the solar system set or as custom setups). These also exhibit expected gravitational dynamics, allowing for the study of simpler interaction patterns.
- **Chaotic Configurations:** By setting up a scenario with Jupiter as the central massive body and placing the other 8 solar system planets in relatively close and dynamically unoptimized orbits around it, we observed the onset of chaotic behavior. Orbits in this configuration quickly deviate from simple ellipses, and bodies can experience significant scattering or ejections over relatively short simulated timescales (e.g., 1-5 years). This demonstrates the simulator's capacity to capture the complex and sensitive nature of certain N-body problems. (NOTE: And also confirms that our System can handle mass-relations that do not yield near UCM behaviors).

These initial 2D demonstrations validate the core N-body Hamiltonian physics implemented in Loma and its successful differentiation via `fwd_diff` to drive the simulation using a symplectic integrator. The ability to configure different scenarios with varying numbers of bodies (up to `MAX_N_BODIES_CONST = 20`) from the Python host showcases the flexibility of the compiled Loma module. This confirms the viability of our baseline goal: implementing a simple physics simulation in the context of Planetary Motion utilizing Loma's unique AD features.

4 Final Project Deliverables

The current implementation of LomaVerse provides a foundational N-body simulator utilizing Hamiltonian mechanics and Loma's automatic differentiation. As outlined, future development will concentrate on expanding its application to complex astrodynamics problems, enhancing its usability and visualization capabilities, exploring advanced numerical methods, and rigorously validating its results. We need to confirm its robustness across different testing scenarios and ultimately assess the efficacy of the differentiable programming approach for these tasks.

4.1 Spacecraft Trajectory Optimization using Reverse Mode AD

At this point, we have a functional GUI to show the validity of our Loma simulation. For our final project, a primary goal is to extend LomaVerse for spacecraft trajectory optimization, such as determining a fuel-optimal path to a target celestial body. This involves leveraging the numerical methods described (Symplectic Euler for N-body propagation) and Loma's reverse mode automatic differentiation.

Problem Formulation. The spacecraft will be integrated as an $(N+1)^{th}$ body, subject to gravitational forces from the other N bodies and a controllable thrust vector $\mathbf{T}(t)$. Its equations of motion will be:

$$\dot{\mathbf{r}}_s = \mathbf{v}_s$$

$$\dot{\mathbf{v}}_s = \sum_{j \neq s}^{N} G \frac{m_j}{||\mathbf{r}_j - \mathbf{r}_s||_{soft}^3} (\mathbf{r}_j - \mathbf{r}_s) + \frac{\mathbf{T}(t)}{m_s(t)}$$

where $m_s(t)$ is the spacecraft's time-varying mass, decreasing due to fuel consumption, modeled by $\dot{m}_s = -||\mathbf{T}(t)||/(I_{sp}g_0)$ (I_{sp} being specific impulse). The control parameters will be the components of $\mathbf{T}(t)$, possibly discretized over time segments.

Loss Function and Optimization. A scalar loss function L will quantify the mission objective. For instance, a rendezvous mission could aim to minimize a weighted sum of final position and velocity errors relative to the target, plus a term for fuel consumption (or total ΔV):

$$L = w_1||\mathbf{r}_s(T_f) - \mathbf{r}_{\text{target}}(T_f)||^2 + w_2||\mathbf{v}_s(T_f) - \mathbf{v}_{\text{target}}(T_f)||^2 + w_3 \int_0^{T_f} ||\mathbf{T}(t)||dt$$

The entire simulation, which propagates the N-body system including the spacecraft under a given thrust profile and computes L, becomes a differentiable function of the thrust control parameters. Loma's `reverse_diff` capability will be applied to this function to compute the gradient $\nabla_{\text{control\_params}} L$ efficiently.

4.2 GUI Connection and User Interface Enhancement

To improve usability and interaction:

- **GUI Integration:** The existing Python host, currently using `matplotlib` for visualization, will be more deeply integrated with a dedicated GUI framework, potentially PyGame as initially discussed by team members. This will allow for more responsive controls and custom visual elements.
- **User-Friendly Interface:** Develop controls for easy scenario selection (Solar System, Jupiter-centered, custom setups), real-time modification of key simulation parameters (e.g., Δt, ϵ), and intuitive start/-pause/reset/step functionalities.
- **Data Display:** Incorporate real-time display of important physical quantities (e.g., total system energy, specific orbital elements of a selected body) within the GUI.

4.3 3D Simulations and Visualizations

Extending the simulation and visualization to three dimensions is a natural progression:

- **3D Physics:** The Loma physics engine (`planetary_motion.py`) will be upgraded. `Vec2` structures for position and momentum will become `Vec3`. The Hamiltonian and its derivatives will incorporate the z-component, requiring modifications to functions like `n_body_hamiltonian` and `get_dH_dr_k_alpha`.
- **3D Visualization:** The Python host will need to employ 3D plotting libraries. While `matplotlib`'s `mplot3d` can provide basic 3D views, libraries like VisPy will be explored for enhanced performance, interactivity, and for higher accuracy simulations.

4.4 Validation and Refinement of Trajectory Optimization

Once the trajectory optimization framework is operational:

- **Validation:** Optimized trajectories will be validated against known solutions (e.g., Earth-Mars transfers, lunar flybys). Conservation of physical quantities (energy, angular momentum) specific to the spacecraft's trajectory under thrust will also be monitored.
- **Loss Function Refinement for `reverse_diff`:** The definition of loss functions L will be critical. This includes careful selection of weighting factors, incorporation of path constraints (e.g., maximum thrust, closest approach distances) as penalty terms or using constrained optimization techniques, and ensuring the loss landscape is amenable to gradient-based methods. The interaction between the chosen loss function and the `reverse_diff` process in Loma will be a key area of investigation to ensure accurate and stable gradient computations.

These future steps aim to evolve LomaVerse into a powerful and validated tool for both simulating complex gravitational dynamics and solving practical astrodynamics optimization problems.