Differentiable N-body code for Galactic Dynamics Odisseo

Giuseppe Viterbo[1] **Tobias Buck**[1]
[1]Interdisciplinary Center for Scientific Computing, Heidelberg University, Germany
giuseppe.viterbo@iwr.uni-heidelberg.de
tobias.buck@iwr.uni-heidelberg.de

Abstract

We introduce ODISSEO (Optimized Differentiable Integrator for Stellar Systems Evolution of Orbits), a differentiable N-body code designed to constrain the gravitational potential of the Milky Way (MW) through dynamical modeling of accreted structures such as stellar streams. ODISSEO is implemented in JAX, enabling just-in-time compilation, automatic differentiation, and hardware acceleration on GPUs and TPUs. The code features efficient, fully vectorized force calculations and exhibits near-linear scaling when distributing a single simulation across multiple GPUs, making it suitable for large-scale optimization tasks. As a demonstration, we present a case study using a mock GD-1 stellar stream simulation, where we optimize four physical parameters via gradient descent: the accretion time and progenitor mass, as well as the masses of the host galaxy's Navarro-Frenk-White (NFW) halo and Miyamoto–Nagai (MN) disk. ODISSEO accurately recovers stream morphology and underlying parameters in a differentiable and scalable framework, providing a powerful tool for dynamical studies of the Milky Way and its accreted substructures. The code is available on GitHub: https://github.com/vepe99/Odisseo.

1 Motivation and Related Work

Stellar streams In the hierarchical galaxy-formation paradigm, large galaxies like the Milky Way are expected to host many stellar streams – long, thin tidal debris from disrupted dwarf galaxies and globular clusters [2]. As orbiting satellites lose stars to tidal forces, they create dynamically cold stellar streams that roughly trace the progenitor's orbit. These streams remain coherent over billions of years, making them extremely sensitive to even small perturbations by the Galactic potential or dark substructures. These features establish stellar streams as powerful probes of the Milky Way's gravitational field and dark-matter distribution.

N-body codes Our motivation in developing ODISSEO was to combine the expressivity of full N-body simulations with the differentiability of modern machine-learning–oriented approaches. Established codes such as PeTar [12], NBODY6++GPU [13], and GADGET-4 [9] provide state-of-the-art accuracy and performance for large-scale stellar dynamics, but they lack native support for automatic differentiation and gradient calculations. On the other hand, recent differentiable semi-analytic models of stellar streams [1, 7], demonstrate the power of gradient-based inference, but they necessarily approximate the underlying dynamics. ODISSEO bridges these paradigms by implementing a fully differentiable direct N-body solver in JAX, enabling gradients of the final particle state with respect to initial conditions and potential parameters, while also inheriting JAX's just-in-time compilation and native GPU acceleration. At present, ODISSEO uses only direct pairwise force computations with softening, without a tree or fast-multipole algorithm. This ensures exact

1st Workshop on Differentiable Systems and Scientific Machine Learning@ EurIPS 2025.

forces (up to the softening scale), which is advantageous for accuracy and gradient consistency, but comes at the cost of computational scalability, making large-N ($> 10^6$) simulations impractical.

2 Odisseo

ODISSEO is designed to enable inference-driven studies of dynamical systems. A key design choice in ODISSEO is to rely on the Python programming language, in contrast to established N-body codes described in Sec 1, which are implemented in low-level languages like C or C++. While these mature codes achieve exceptional performance, their complexity and low-level implementations often make rapid prototyping, unit testing, and community contributions more difficult. By building ODISSEO entirely in Python, we prioritize maintainability, readability, and accessibility for a community developments. This choice facilitates faster iteration on scientific ideas, easier integration with modern ML workflows, and a lower barrier to entry for community-driven extensions. Written in a purely functional style and built entirely on the JAX ecosystem, ODISSEO provides end-to-end differentiability: the final particle state remains differentiable with respect to all simulation parameters, including initial conditions, integration time, particle masses, and external potential parameters. This is achieved by leveraging JAX's program transformation capabilities, such as tracing and reverse-mode automatic differentiation, which propagate gradients through the entire simulation pipeline. The functional and modular design, inspired by [10], allows components—such as integrators, external potentials, or initial condition generators—to be easily swapped or extended, facilitating rapid experimentation. ODISSEO also natively supports just-in-time compilation, automatic vectorization, and execution across CPUs, GPUs, and TPUs, ensuring both flexibility and high performance. Looking forward, ODISSEO aims to serve as a community-driven platform for differentiable simulations, adaptable to astrophysical problems such as stellar stream modeling, as well as to other domains where dynamical systems and external potentials play a central role.

2.1 Distributed performance

To demonstrate that ODISSEO is ready for large-scale parallelization with minimal user effort, we benchmarked it on a standard stellar-dynamical test: the tidal disruption of a Plummer sphere in a realistic Milky Way potential. Leveraging the JAX ecosystem, which provides built-in primitives for distributed data and computation, we distributed a single simulation run across multiple GPUs (NVIDIA A100) without modifying the simulation code. Specifically, we ran the same disruption experiment with increasing particle numbers while distributing the computation over 1 (baseline), 2, 3, 4, and 5 GPUs to measure scaling performance. The results are shown in Fig. 1. Each configuration was repeated three times to report both the mean runtime and its standard deviation. This setup highlights the ease with which ODISSEO achieves near-linear strong scaling on modern GPU clusters, confirming its suitability for massively parallel inference-driven simulations.

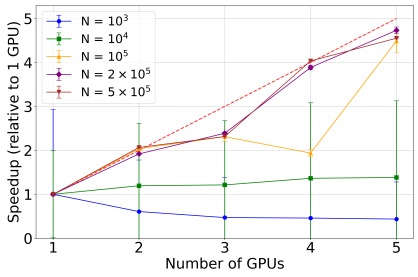

Figure 1: Speedups for a single simulation distributed across multiple GPUs with different number of particles N. The dashed red line is the ideal linear scale, which is achieved for N > 10^4.

2.2 Inference

We demonstrate two applications of gradient-based optimization for studying stellar streams with a differentiable N-body code. Importantly, we do not work with real GD-1 observations, but with a mock simulated stream for which the true parameters are known. The first method, described in Sec. 2.2.1, introduces a novel approach that treats both observational and simulated data as point clouds and minimizes their discrepancy directly. The second method, inspired by previous work on stellar streams [5], assumes that stream stars follow orbits similar to that of their progenitor. This allows the inverse problem to be reformulated as a least-squares minimization between the observed stars and the simulated orbit.

2.2.1 Gradient Descent and Bootstrapping

To illustrate a simple inference example using our differentiable N-body simulator, we use the full 6D phase-space information of particles to recover the parameters Θ that best describe the

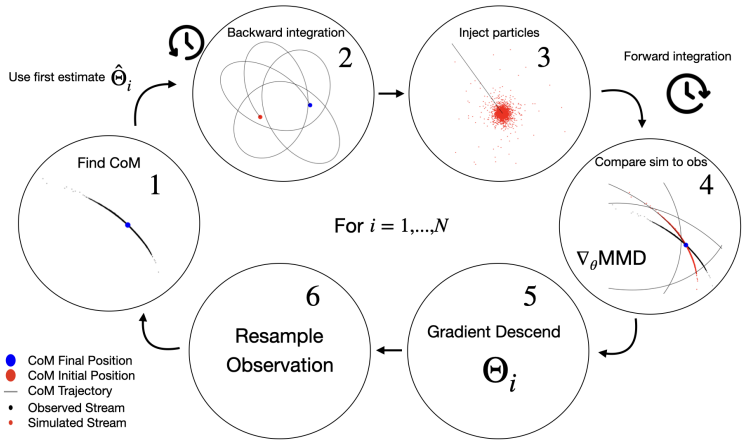

Figure 2: Inference pipeline. The goal is to generate N bootstrap samples of the parameters. 1. From the observed stellar stream, we identify its Center of Mass (CoM). 2. Using an initial estimate $\hat{\Theta}_i = (\hat{t}_{\rm end}, \hat{M}_{\rm Plummer}, \hat{M}_{\rm NFW}, \hat{M}_{\rm MN})$, obtained through a grid search, we integrate the present-day CoM position backward in time as a single particle of mass $\hat{M}_{\rm Plummer}$ for a duration $\hat{t}_{\rm end}$, within an external potential parametrized by $\hat{M}_{\rm NFW}, \hat{M}_{\rm MN}$. 3. The recovered initial CoM position is populated with 1000 particles sampled from a Plummer sphere in equilibrium. 4. The resulting simulated stream is compared to the observed stream using the Maximum Mean Discrepancy (MMD), and the gradient of the loss is computed. 5. We perform 10 steps of vanilla Gradient Descent, repeating steps 2–4 internally. The parameters at the minimum MMD are taken as the final estimate for the sample $\Theta_i = (t_{\rm end}, M_{\rm Plummer}, M_{\rm NFW}, M_{\rm MN})$. 6. The observed stream is resampled according to observational errors to generate a new realization of the target, and the process is repeated from step 1.

host galaxy potential—specifically, the masses of the NFW halo and the MN disk—as well as the progenitor properties, including the Plummer sphere mass and the total integration time, which are respectively $\Theta = (t_{\rm end}, M_{\rm Plummer}, M_{\rm NFW}, M_{\rm MN})$. Following the general setup of [1], we adopt the `MWPotential2014` model from `galpy` [3] as the analytic Milky Way potential.

The overall inference pipeline is summarized in Fig. 2. Starting from the present-day position and velocity of the stream's progenitor, we integrate its center of mass backward in time as a single particle to obtain the initial conditions for the simulation. Unlike the semi-analytic model of [1], the progenitor is then replaced by 1000 particles drawn from an equilibrium Plummer sphere, shifted in phase space by the backward-integrated initial conditions. The system is evolved forward in time using a second-order Velocity–Verlet integrator with 1000 fixed time steps. A few representative snapshots and the progenitor orbit are shown in Fig. 5. For the inference, we employ a bootstrapping procedure: the simulated GD-1 stream used as "observation" is resampled 1000 times according to the observational uncertainties reported in [1], and the inference is repeated for each realization. The pipeline consists of two stages: first, a grid search over 10^4 parameter combinations using the Maximum Mean Discrepancy (MMD) between simulated and observed streams as the loss function; second, 10 refinement steps of gradient-based optimization with a learning rate of 10^{-5} to improve upon the grid-search results. Finally, the distribution obtained by bootstrapping is summarized by fitting a kernel density estimate (KDE) to the inferred parameters, from which we generate the corner plot in Fig. 3. The experiment was run on 2 NVIDIA A100. The code to reproduce the results is available on GitHub: `https://github.com/vepe99/Odisseo/blob/main/notebooks/dev/albastross/uncertainty_quantification.py`

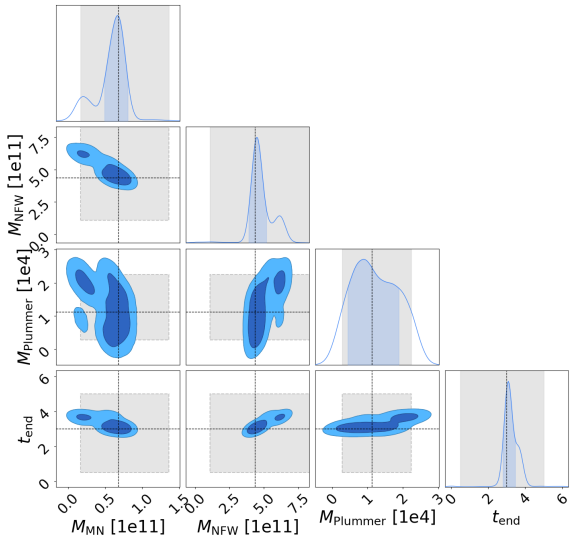

Figure 3: Bootstrapping inference results. The masses $M_{\rm MN}$, $M_{\rm NFW}$ and $M_{\rm Plummer}$ are in $M_\odot$, while the total time of integration $t_{\rm end}$ is in Gyr. The gray box represents the "prior" used to set up the initial grid search. The darker area in the diagonal plots represents one standard deviation from the maximum likelihood. The dashed lines indicate the true values.

2.2.2 Orbit fitting and Fisher contours

For this method, we evolve only a single particle–the progenitor– with mass $M_{\rm Plummer} = 10^{4.5}\,{\rm M}_\odot$ and compare its orbit with the observed stream, following the setup described in [6]. The integration is both forward and backward in time for $\Delta t = 2\,{\rm Gyr}$, starting from its present-day position.[1] This integration yields an orbit $(\mathbf{X}(t), \mathbf{V}(t))$, which we project onto the stream's reference frame defined by $(\phi_1, \phi_2, \tilde{\mu}_{\phi_1}, \mu_{\phi_2}, v_h)$, introduced by [5]. In this frame, ϕ_1 and ϕ_2, corresponding respectively to the longitude and latitude in the GD-1 celestial coordinate system, are expressed in radians, $\tilde{\mu}_{\phi_1} = \mu_{\phi_1}\cos\phi_2$, and μ_{ϕ_2}, are proper motions in ${\rm mas\,yr^{-1}}$, and v_h is the heliocentric line-of-sight velocity in ${\rm km\,s^{-1}}$. Following the same observational window and data treatment as in [6], we interpolate the modeled orbit along ϕ_1 to match the sampling of the observed stream. The goodness of fit is then quantified through a chi-squared statistic, and adopting the same error values as [6]. The resulting χ^2 serves as the log-likelihood, $\mathcal{L} \propto -\chi^2$, for the parameter inference. To obtain the Maximum Likelihood Estimate (MLE), we apply the Levenberg–Marquardt least-squares method implemented in `optimistix` [8] from 1000 randomly sampled initial positions θ_i (with $0.5\,\theta_{True} < \theta_i < 2\theta_{True}$) and then take the value with the lowest χ^2. The confidence intervals are derived from the inverse of the Fisher information matrix, assuming a Gaussian approximation of the likelihood around the MLE with fixed covariance matrix. The Fisher matrix is defined as $F_{ij} = \frac{1}{2}\frac{\partial^2\chi^2}{\partial\theta_i\,\partial\theta_j}$, where θ_i and θ_j denote the model parameters, in our case the NFW virial mass M_{vir}^{NFW} and scale radius r_s^{NFW}, the MN mass M^{MN}, scale height b^{MN} and length a^{MN}. The Fisher matrix is trivially obtained using `jax.hessian`. The covariance matrix of the estimated parameters is then given by $C = F^{-1}$. The results are reported in Fig. 4. The code to reproduce the results is available on GitHub: `https://github.com/vepe99/Odisseo/blob/main/notebooks/dev/albastross/Fisher_contour/Least_square/least_square_search_paper.py`.

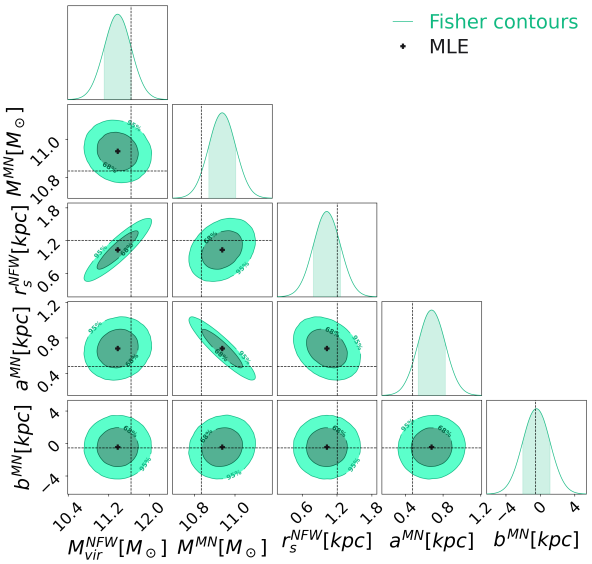

Figure 4: Fisher confidence contours around the Maximum Likelihood Estimate (MLE). All parameter values are shown in logarithmic (dex) units. The dashed lines are indicating the position of the true value.

3 Conclusion and outlook

We have presented ODISSEO, a novel, modular, highly parallelizable, differentiable direct N-body code written in JAX, and demonstrated that even complex tasks—such as retrieving the parameters of the host galaxy and the progenitor of a stellar stream—can be tackled with surprisingly high accuracy. Given the computationally demanding nature of the grid search used in our first example, this inference pipeline should be considered primarily as a proof of concept rather than a fully deployable application. The second approach is strongly limited by the Gaussian approximation and by the non-negligible dependency of the MLE algorithm on the initial guess θ_i. A complete analysis would require a proper treatment of uncertainties, for instance via Hamiltonian Monte Carlo or simulation-based inference with simulator feedback, as suggested in [4]. Also as future works, we aim to leverage the gradient in order to couple the ODISSEO with a Neural Network in order to improve the accuracy of the solver, just like as been shown in [11].

[1] For the GD-1 stream, this assumption does not strictly hold since the progenitor is fully dissolved. However, in principle, the initial phase-space coordinates could be included as additional free parameters to infer.

Broader impact statement

The authors are not aware of any immediate ethical or societal implications of this work. This work purely aims to aid scientific research and proposes to apply novel differentiable simulator to unravel the formation history of the Milky Way, its morphology and the family of stellar streams that populate its halo.

Acknowledgments and Disclosure of Funding

This work is funded by the Carl-Zeiss-Stiftung through the NEXUS program. This work was supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy EXC 2181/1 - 390900948 (the Heidelberg STRUCTURES Excellence Cluster). We acknowledge the usage of the AI-clusters Tom and Jerry funded by the Field of Focus 2 of Heidelberg University.

References

[1] James Alvey, Mathis Gerdes, and Christoph Weniger. Albatross: A scalable simulation-based inference pipeline for analysing stellar streams in the milky way.

[2] Ana Bonaca and Adrian M. Price-Whelan. Stellar streams in the gaia era.

[3] Jo Bovy. galpy: A python library for galactic dynamics. 216(2):29.

[4] Benjamin Holzschuh and Nils Thuerey. Flow matching for posterior inference with simulator feedback.

[5] Sergey E. Koposov, Hans-Walter Rix, and David W. Hogg. CONSTRAINING THE MILKY WAY POTENTIAL WITH A SIX-DIMENSIONAL PHASE-SPACE MAP OF THE GD-1 STELLAR STREAM. *The Astrophysical Journal*, 712(1):260–273, March 2010.

[6] Martín Federico Mestre, Carlos Raul Argüelles, Daniel Diego Carpintero, Valentina Crespi, and Andreas Krut. Modeling the track of the GD-1 stellar stream inside a host with a fermionic dark matter core-halo distribution. , 689:A194, September 2024.

[7] Jacob Nibauer, Ana Bonaca, David N. Spergel, Adrian M. Price-Whelan, Jenny E. Greene, Nathaniel Starkman, and Kathryn V. Johnston. `StreamSculptor`: Hamiltonian perturbation theory for stellar streams in flexible potentials with differentiable simulations.

[8] Jason Rader, Terry Lyons, and Patrick Kidger. Optimistix: modular optimisation in jax and equinox. *arXiv:2402.09983*, 2024.

[9] Volker Springel, Rüdiger Pakmor, Oliver Zier, and Martin Reinecke. Simulating cosmic structure formation with the gadget-4 code. 506(2):2871–2949.

[10] Leonard Storcks and Tobias Buck. Differentiable conservative radially symmetric fluid simulations and stellar winds – jf1uids.

[11] Kiwon Um, Robert Brand, Yun, Fei, Philipp Holl, and Nils Thuerey. Solver-in-the-loop: Learning from differentiable physics to interact with iterative PDE-solvers.

[12] Long Wang, Masaki Iwasawa, Keigo Nitadori, and Junichiro Makino. PeTar: a high-performance n-body code for modeling massive collisional stellar systems. 497(1):536–555.

[13] Long Wang, Rainer Spurzem, Sverre Aarseth, Keigo Nitadori, Peter Berczik, M. B. N. Kouwenhoven, and Thorsten Naab. nbody6++gpu: ready for the gravitational million-body problem. 450(4):4070–4080.

A Simulation Output

In this Fig. 5 we show a few time steps for the fiducial simulation of the GD1 stream.

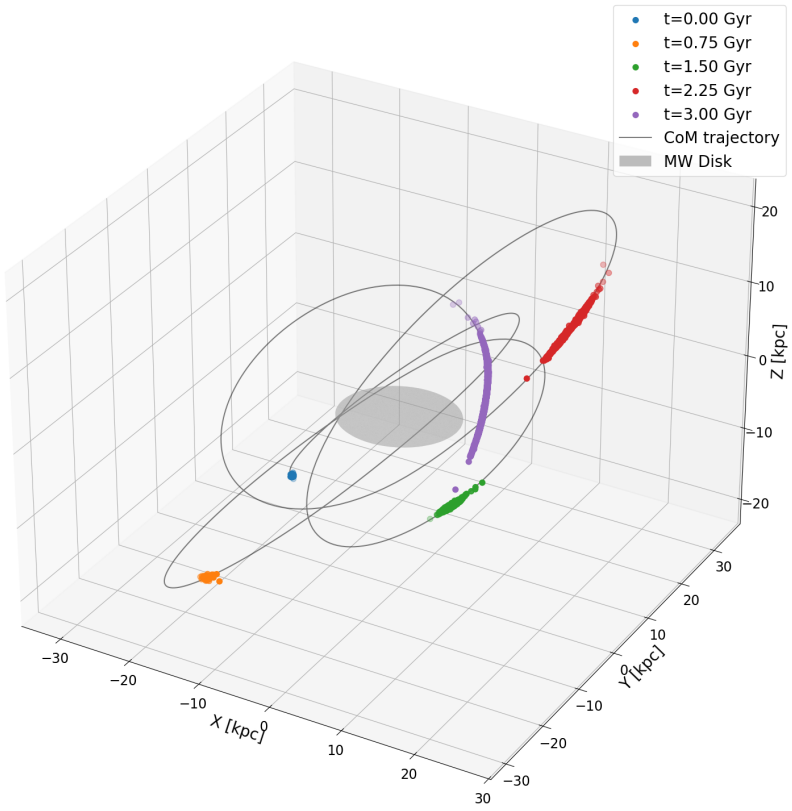

Figure 5: Snapshots at different times t of the GD1 stream simulation. Milky Way's stellar disk is shown for scale and the center of mass (CoM) trajectory is shown as solid line.