Geodesic Integration in Julia

Patrick Anker

New York University

(Dated: May 11, 2018)

Abstract

In this article I present a new toolkit for tracing particle trajectories through spacetime in Julia. I discuss the differences between direct and ADM integration, their benefits and drawbacks. Finally, I show that the direct method is highly accurate with a manifold projection technique to conserve energy and angular momentum, when the metric admits those to be true. Example orbits can be found in an attached IPython Notebook file that allows interactivity.

CONTENTS

I. INTRODUCTION

Geodesic integration comes in two flavors in numerical relativity: using the direct geodesic equation or using the ADM formalism. The easiest method is from the geodesic equation

$$\frac{\mathrm{d}u^\alpha}{\mathrm{d}\tau} = -\Gamma^\alpha_{\beta\gamma} u^\beta u^\gamma \tag{I.1}$$

which allows direct integration with respect to proper time τ (or some affine parameter λ for lightlike particles). This is a series of four non-linear, coupled differential equations which would be heinous to solve in the analytic scheme, except for specific cases. The benefit of using the geodesic equation directly is that it's possible to use direct numerical ODE methods like RK4 to solve the trajectories of particles. The downside is that the metrics have to be analytically known, which limits the capabilities of a general geodesic raytracer. On its own, the metric cannot be numerically determined using 4D geometry alone. It's necessary, then, to split the timelike component from the spacelike components in spacelike foliations $\{\Sigma_t\}$ of constant t. This formalism, the ADM or 3+1 formalism, is more general for metrics that are not numerically determined. The 3+1 split is a Hamiltonian approach that allows for symplectic integration using

$$\frac{\mathrm{d}x^i}{\mathrm{d}t} = \frac{\partial \mathcal{H}}{\partial u_i} = \gamma^{ij}\frac{u_j}{u^0} - \beta^i \tag{I.2}$$

$$\frac{\mathrm{d}u_i}{\mathrm{d}t} = -\frac{\partial \mathcal{H}}{\partial x^i} = -\alpha u^0 \partial_i \alpha + u_k \partial_i \beta^k - \frac{u_j u_k}{2u^0}\partial_i \gamma^{jk} \tag{I.3}$$

This report uses the terminology found in [1], which calls α the "lapse function" and β^i the "shift vector", the definitions of which are given in the 4-metric

$$g_{\mu\nu} = \begin{pmatrix} -\alpha^2 + \beta_k\beta^k & \beta_i \\ \beta_j & \gamma_{ij} \end{pmatrix}$$

where $\gamma_{ij} = g_{ij}$. The latin indices indicate the spatial coordinates.

II. METHODS

Writing a solver for geodesics in numerically determined spacetimes is not a trivial issue as you would need to first evolve the metric in a foliated hypersurface Σ_t and then determine the next geodesic step using the evolved metric. Instead, I first tried to implement a symplectic routine using the ADM formalism for a known, energy-conserving metric. Orbits generally require energy and angular momentum conservation to prevent numerical loss. As such, the symplectic methods — like Verlet leapfrog — prove useful for these situations. Thankfully, most of the metrics we studied this year in the proximity of black holes are energy-conserving, so I didn't have to worry about a non-conservative black hole implementation.

The issue arose in the implementation of the gradient operation. Because I was using analytic metrics, I wanted to use an analytic implementation of the gradient as opposed to a discretized "stamp" operation on a mesh. It proved a little tricky to implement this correctly, since the distinction between 3-vectors and 4-vectors is subtle for a machine. The metric is evaluated at a 4-vector, but the gradients expect a 3-vector. I ultimately solved this issue, but I moved on to implement the direct geodesic integrator. The backend for the integration routines is DifferentialEquations.jl which provides a plethora of numerical routines and methods, including the Tsit5 algorithm.[2][3] The Tsit5 algorithm is a quicker version of the Dormand-Prince RK4 routine.

To prevent energy or angular momentum loss while not using a symplectic method for integration, DifferentialEquations.jl offers a callback system that allows manifold projections, which constrain particular parameters after each time step given some Jacobian or some system of equations. Rather than using a Hamiltonian manifold projection ($\mathcal{H}(x^\mu, u_\mu) - E = 0$), I instead used any Killing vectors as constraints e.g. e and l from $-\vec{\xi} \cdot \vec{u}$ and $\vec{\eta} \cdot \vec{u}$, respectively. At the start of each integration run, the module GeodesicEquations.jl finds

the conserved quantities and stores them in a function that calculates the residuals in the manifold projection routine. For e (for example), each step would trigger this calculation:

$$r(x^\mu, u^\mu) = -g_{0\mu}u^\mu + e$$

If the residual was larger than a tolerance level, DifferentialEquations.jl would minimize the residuals with respect to a Jacobian. There is a computational slowdown to this method, but it allows long proper time integrations ($\sim\ 10^8$).

III. ORBITS

The most simple orbit, the ISCO for a Schwarzschild black hole, was used as the test case for the algorithm. For massive particles, the ISCO of black hole is

$$r_{\text{ISCO}} = 6M$$

which corresponds to $l/M = \sqrt{12}$. Setting the initial angular velocity to be the appropriate l with the initial position at $\{r = 6M, \theta = \pi/2, \phi = 0\}$, I compared the numerically integrated orbit with respect to the theoretical model. The M chosen for all orbits is $M = 6.0$.

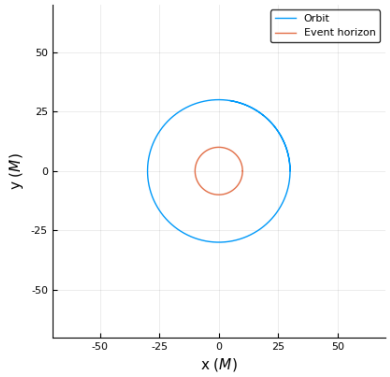

FIG. 1. A wonderfully circular orbit at 6M

The nicely circular orbit in Fig. 1 pairs nicely with the relative error in the orbit with respect to the theory: of order 10^{-15}! This is roughly the best one could get in terms of numerical errors. Fig. 2 shows the trend in error increase over time.

The rest of the orbits that I computed can be found in the attached "orbits.ipynb" file.

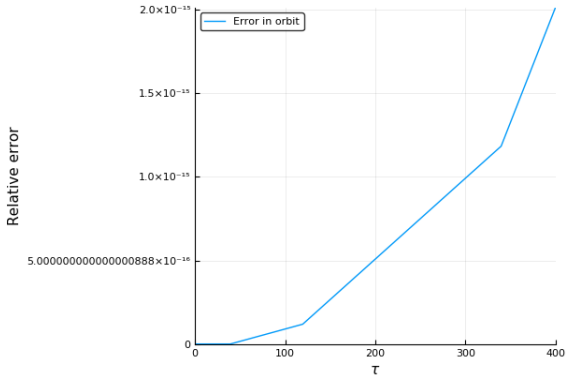

FIG. 2. Error in circular orbit

IV. DISCUSSION

I wish I had started earlier or had more time to correctly implement the ADM formalism to compare the symplectic and forward integration schemes and how the manifold projection algorithm compares in accuracy with respect to the symplectic energy conservation. This was my first goal with this project, but I did not have the time to completely develop the idea. Furthermore, expanding the timelike integration part to include Doppler shift would be an essential addition to create a ray tracer, like GYOTO.[4] However, the package is useful as it is for an example for how to implement geodesic integration in Julia. I may continue tinkering with the module in the future!

---

[1] F. Bacchini, B. Ripperda, A. Yuran Chen, and L. Sironi, ArXiv e-prints (2018), arXiv:1801.02378 [astro-ph.HE].

[2] C. Rackauckas and Q. Nie, Journal of Open Research Software **5** (2017), 10.5334/jors.151/.

[3] C. Tsitouras, Computers & Mathematics with Applications **62** (2011), 10.1016/j.camwa.2011.06.002.

[4] F. H. Vincent, T. Paumard, E. Gourgoulhon, and G. Perrin, Classical and Quantum Gravity **28**, 225011 (2011), arXiv:1109.4769 [gr-qc].