The Journal of Open Source Software

pytreegrav: A fast Python gravity solver

Michael Y. Grudić[1] **and Alexander B. Gurvich**[1]

1 Department of Physics & Astronomy and CIERA, Northwestern University, 1800 Sherman Ave, Evanston, IL 60201, USA

DOI: 10.21105/joss.03406

Software

- Review
- Repository
- Archive

Editor: Pending Editor

Submitted: 22 June 2021
Published: 25 June 2021

Summary

Gravity is important in a wide variety of science problems. In particular, questions in astrophysics nearly all involve gravity, and can have large ($>> 10^4$) numbers of gravitating masses, such as the stars in a cluster or galaxy, or the discrete fluid elements in a hydrodynamics simulation. Often the gravitational field of such a large number of masses can be too computationally expensive to compute by directly summing the contribution of every single element at every point of interest.

`pytreegrav` is a multi-method Python package for computing gravitational fields and potentials. It includes an exact direct-summation ("brute force") solver and a fast, approximate tree-based method that can be orders of magnitude faster than the naïve method. It can compute fields and potentials from arbitrary particle distributions at arbitrary points, with arbitrary softening/smoothing lengths, and is parallelized with OpenMP.

Statement of need

The problem addressed by `pytreegrav` is the following: given an arbitrary set of "source" masses m_i with 3D coordinates $\mathbf{x}_i$, and optionally each having a finite spatial extent h_i (the *softening radius*), one would like to compute the gravitational potential Φ and/or the gravitational field $\mathbf{g}$ at an arbitrary set of "target" points in space $\mathbf{y}_i$. A common application for this is N-body simulations (wherein $\mathbf{y}_i = \mathbf{x}_i$). It is also often useful for *analyzing* simulation results after the fact – Φ and $\mathbf{g}$ are sometimes not saved in simulation outputs, and even when they are it is often useful to analyze the gravitational interactions between specific *subsets* of the mass elements in the simulation. Computing $\mathbf{g}$ is also important for generating equilibrium *initial conditions* for N-body simulations (Volker Springel & White, 1999; Yurin & Springel, 2014), and for identifying interesting gravitationally-bound structures such as halos, star clusters, and giant molecular clouds (Behroozi et al., 2013; Grudić et al., 2018; Guszejnov et al., 2020).

Many gravity simulation codes (or multi-physics simulation codes *including* gravity) have been written that address the problem of gravity computation in a variety of ways for their own internal purposes (Aarseth, 2003; Dehnen & Read, 2011). However, `pykdgrav` (the precursor of `pytreegrav`) was the first Python package to offer a generic, modular, trivially-installable gravity solver that could be easily integrated into any other Python code, using the fast, approximate tree-based Barnes & Hut (1986) method to be practical for large particle numbers. `pykdgrav` used a KD-tree implementation accelerated with `numba` (Lam et al., 2015) to achieve high performance in the potential/field evaluation, however the prerequisite tree-building step had relatively high overhead and a very large memory footprint, because the entire dataset was redundantly stored at every level in the tree hierarchy. This made it difficult to scale to various practical research problems, such as analyzing high-resolution

galaxy simulations (Gurvich et al., 2020). `pytreegrav` is a full refactor of `pykdgrav` that addresses these shortcomings with a new octree implementation, with drastically reduced tree-build time and memory footprint, and a more efficient non-recursive tree traversal for field summation. This makes it suitable for post-processing datasets from state-of-the-art astrophysics simulations, with upwards of 10^8 particles in the region of interest.

Methods

`pytreegrav` can compute Φ and $\mathbf{g}$ using one of two methods: by "brute force" (explcitly summing the field of every particle, which is exact to machine precision), or using the fast, approximate Barnes & Hut (1986) tree-based method (which is approximate, but much faster for large particle numbers). In an N-body problem where the fields at all particle positions must be known, the cost of the brute-force method scales $\propto N^2$, while the cost of the tree-based method scales less steeply, $\propto N \log N$ (Figure 1).

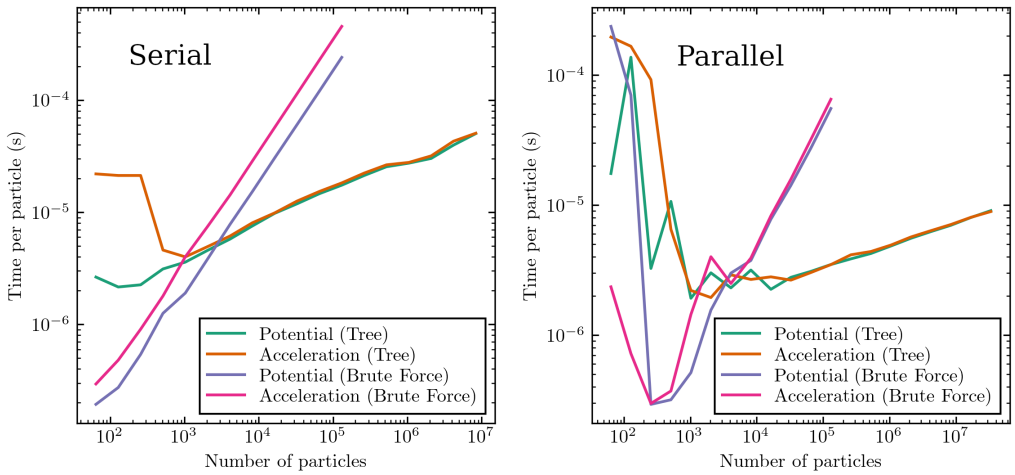

Figure 1: Wall-clock time per particle running `pytreegrav` on a sample of N particles from a Plummer (1911) distribution for various N. Test was run on a an Intel i9 9900K workstation on a single core (*left*) and in parallel on 16 logical cores (*right*).

The brute-force methods are often fastest for small ($< 10^3$ particle) point sets because they lack the overheads of tree construction and traversal, while the tree-based methods will typically be faster for larger datasets because they reduce the number of floating-point operations required. Both methods are optimized with the `numba` LLVM JIT compiler (Lam et al., 2015), and the basic `Accel` and `Potential` front-end functions will automatically choose the method is likely to be faster, based on this heuristic crossover point of 10^3 particles. Both methods can also optionally be parallelized with OpenMP, via the `numba @njit(parallel=True)` interface.

The implementation of the tree build and tree-based field summation largely follows that of `GADGET-2` (V. Springel, 2005). Starting with an initial cube enclosing all particles, particles are inserted into the tree one at a time. Nodes are divided into 8 subnodes until each subnode contains at most one particle. The indices of the 8 subnodes of each node are stored for an initial recursive traversal of the completed tree, but an optimized tree traversal only needs to know the *first* subnode (if the node is to be refined) and the index of the next branch of the tree (if the field due to the node is summed directly), so these indices are recorded in the initial recursive tree traversal, and the 8 explicit subnode indices are then deleted, saving memory and removing any empty nodes from consideration. Once these "next branch" and "first

subnode" indices are known, the tree field summations can be done in a single `while` loop with no recursive function calls, which generally improves performance and memory usage.

The field summation itself uses the Barnes & Hut (1986) geometric opening criterion, with improvements suggested by Dubinski (1996): for a node of side length L with centre of mass located at distance r from the target point, its contribution is summed using the monopole approximation (treating the whole node as a point mass) only if $r > L/\Theta + \delta$, where $\Theta = 0.7$ by default (giving $\sim 1\%$ RMS error in $\mathbf{g}$), δ is the distance from the node's geometric center to its center of mass. If the conditions for approximation are not satisfied, the node's subnodes are considered in turn, until the field contribution of all mass within the node is summed.

`pytreegrav` supports gravitational softening by assuming the mass distribution of each particle takes the form of a standard M4 cubic spline kernel, which is zero outside of the softening radius h (outside which the field reduces to that of a point mass). Explicit expressions for this form of the softened gravitational potential and field are given in Hopkins (2015). h is allowed to vary from particle to particle, and when summing the field the larger of the source or the target softening is used (symmetrizing the force between overlapping particles). When softenings are nonzero, the largest softening $h_{\rm max}$ of all particles in a node is stored, and a node is always opened in the field summation if $r < 0.6L + \max(h_{\rm target}, h_{\rm max}) + \delta$, where $h_{\rm target}$ is the softening of the target particle where the field is being summed. This ensures that any interactions between physically-overlapping particles are summed directly with the softening kernel.

Acknowledgements

We acknowledge code contributions from Ben Keller and Martin Beroiz, and helpful feedback from Elisa Bortolas, Thorsten García, and GitHub user `herkesg` during the development of `pykdgrav`, which were incorporated into `pytreegrav`.

References

Aarseth, S. J. (2003). *Gravitational N-Body Simulations*.

Barnes, J., & Hut, P. (1986). A hierarchical O(N log N) force-calculation algorithm. *324*(6096), 446–449. https://doi.org/10.1038/324446a0

Behroozi, P. S., Wechsler, R. H., & Wu, H.-Y. (2013). The ROCKSTAR Phase-space Temporal Halo Finder and the Velocity Offsets of Cluster Cores. *762*(2), 109. https://doi.org/10.1088/0004-637X/762/2/109

Dehnen, W., & Read, J. I. (2011). N-body simulations of gravitational dynamics. *European Physical Journal Plus*, *126*, 55. https://doi.org/10.1140/epjp/i2011-11055-3

Dubinski, J. (1996). A parallel tree code. *1*(2), 133–147. https://doi.org/10.1016/S1384-1076(96)00009-7

Grudić, M. Y., Guszejnov, D., Hopkins, P. F., Lamberts, A., Boylan-Kolchin, M., Murray, N., & Schmitz, D. (2018). From the top down and back up again: star cluster structure from hierarchical star formation. *481*(1), 688–702. https://doi.org/10.1093/mnras/sty2303

Gurvich, A. B., Faucher-Giguère, C.-A., Richings, A. J., Hopkins, P. F., Grudić, M. Y., Hafen, Z., Wellons, S., Stern, J., Quataert, E., Chan, T. K., Orr, M. E., Kereš, D., Wetzel, A., Hayward, C. C., Loebman, S. R., & Murray, N. (2020). Pressure balance in the multiphase ISM of cosmologically simulated disc galaxies. *498*(3), 3664–3683. https://doi.org/10.1093/mnras/staa2578

Guszejnov, D., Grudić, M. Y., Offner, S. S. R., Boylan-Kolchin, M., Faucher-Gigère, C.-A., Wetzel, A., Benincasa, S. M., & Loebman, S. (2020). Evolution of giant molecular clouds across cosmic time. *492*(1), 488–502. https://doi.org/10.1093/mnras/stz3527

Hopkins, P. F. (2015). A new class of accurate, mesh-free hydrodynamic simulation methods. *450*, 53–110. https://doi.org/10.1093/mnras/stv195

Lam, S. K., Pitrou, A., & Seibert, S. (2015). Numba: A LLVM-based python JIT compiler. *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*. https://doi.org/10.1145/2833157.2833162

Plummer, H. C. (1911). On the problem of distribution in globular star clusters. *71*, 460–470. https://doi.org/10.1093/mnras/71.5.460

Springel, V. (2005). The cosmological simulation code GADGET-2. *364*, 1105–1134. https://doi.org/10.1111/j.1365-2966.2005.09655.x

Springel, Volker, & White, S. D. M. (1999). Tidal tails in cold dark matter cosmologies. *307*(1), 162–178. https://doi.org/10.1046/j.1365-8711.1999.02613.x

Yurin, D., & Springel, V. (2014). An iterative method for the construction of N-body galaxy models in collisionless equilibrium. *444*(1), 62–79. https://doi.org/10.1093/mnras/stu1421