DISCRETE ELASTIC RODS IN JULIA

JAMES GABBARD

1. Introduction. One-dimensional elastic structure are ubiquitous in nature and occur at a variety of scales, ranging from human hair to cilia in microorganisms. They are also a pathway to the creation of 4D printed structures, which undergo a pre-specified change in shape when exposed to an external stimulus [3]. These can be constructed using a fine lattice of curved elastic filaments, which change their curvature in response to changes in temperature or magnetic field. Simulating such a lattice requires an elastic rod model that incorporates variable rest-curvature and anisotropic cross sections, as well as a method for assembling individual rods into a unified structural model.

Discrete Elastic Rods (DER) is a model that provides these capabilities. Proposed in 2008 as an efficient method for animating hair [2], DER has since been applied to the simulation of viscous threads [1], arrays of cilia [8], soft robotics [9], and a variety of other systems based on 1D elastic objects. The core of DER is the definition discrete framed curves, as well as a method for assigning elastic energy to configurations of these curves. Equilibrium problems can then be posed as a constrained minimization of the elastic energy, while dynamics problems use this elastic energy to define the elastic forces and torques acting on the rod. In either case, the minimization or implicit time integration is the main numerical challenge, which has traditionally been tackled using Newton's method with a sparse, analytically-derived Hessian matrix.

Although there is an active and growing community of researchers that continue to extend and improve DER, there is no consensus on implementation. High performance implementations of the model have been used for to animate hair in the graphics community [2], but these typically specialize to an elastic constitutive law and an isotropic cross section. There have also been two lightly documented C++ implementations released by academic research groups [5, 7]. The goal of this research project is to begin to make the case for a concensus implementation in Julia, which takes advantage of the no-cost abstractions and rich automatic differentiation ecosystem available in the language. To showcase these aspects, the implementation described here highlights a flexible problem-specification system that takes advantage of Julia's dispatch system and code generation capability, as well as the performance of gradient and Hessian calculations based entirely on automatic differentiation.

Section 2 of this work provides an overview of the Discrete Elastic Rods model, synthesizing the work of [2], [1], and [6]. Section 3 gives an overview of the implementation and the problem-specification system, while Section 4 focuses on a performance analysis of Discrete Elastic Rods in conjunction with Julia's forward and reverse mode automatic differentiation packages. Finally, Section 5 concludes by touching on several future directions for DER in Julia.

2. Rod Modeling.

2.1. Kinematics. The centerline of the elastic rod can be represented by a collection of vertices $\{\mathbf{x}_0,\ \mathbf{x}_1,\ ...,\ \mathbf{x}_n\}$, connected in sequence by edge vectors $\mathbf{e}^i = \mathbf{x}_{i+1} - \mathbf{x}_i$. For convenience, all quantities associated with the vertices are given a lower index, while quantities associated with the edges are given an upper index. Each edge $\mathbf{e}^i$ has length $l^i = \left\|\mathbf{e}^i\right\|$, and can be normalized to produce a unit vector $\mathbf{t}^i = \mathbf{e}^i/l^i$ that is tangent to the discrete centerline. It will also be convenient is some calculations to associate a length to each vertex, which is taken to be $l_i = (l^{i-1} + l^i)/2$.

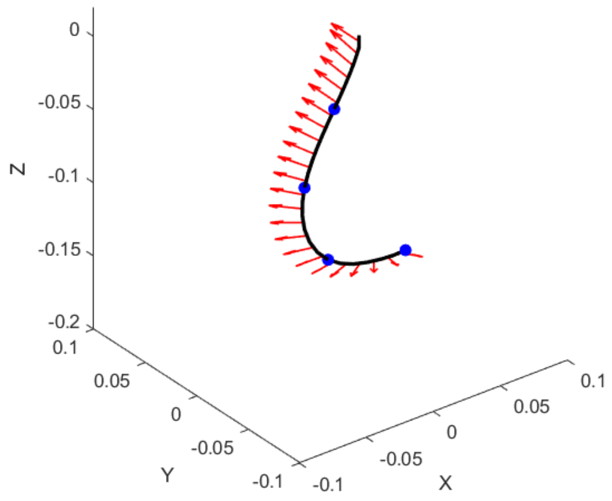


Fig. 1: The centerline $\{\mathbf{x}_i\}$ and directors $\{\mathbf{d}_i\}$ of an elastic rod, along with some gratuitous markings that track the quarter-points of the undeformed centerline.

To complete the kinematic description of an elastic rod, it is necessary to specify the shape and orientation of the cross section on each edge. DER operates under the assumption that a rod is unshearable, indicating that its cross section remains undeformed and perpendicular to the centerline of the rod. However, the cross-section is still free to rotate about the centerline, so that it is necessary to specify the orientation of the cross section on each edge. This is done by specifying a series of unit vectors $\mathbf{d}_1^i$ on each edge which are perpendicular to $\mathbf{t}^i$. The vectors $\mathbf{d}_1^i$ and $\mathbf{d}_2^i = \mathbf{t}^i \times \mathbf{d}_1^i$ are termed directors, and together with $\mathbf{t}^i$ they form an orthonormal triad on each edge of the rod.

The shape of the rod's cross section is represented only by integrated quantities, the cross sectional area A^i and area moment of inertia tensor $\mathbf{I}^i$. The latter is a 2×2 symmetric tensor, and can expressed by three components I_{11}^i, I_{12}^i, and I_{22}^i in the $(\mathbf{d}_1^i,\, \mathbf{d}_2^i)$ frame. The polar moment of inertia $J^i = \operatorname{tr} \mathbf{I}^i$ also plays a role in defining the rod's twisting stiffness. (In the following, $\mathbf{I}$ will be used for both the inertia and identity matrices; the meaning will be clear from the context).

Although the vertices $\{\mathbf{x}_i\}$, directors $\left\{\mathbf{d}^i\right\}$, and sectional properties $\mathbf{A}_i$, $\mathbf{I}^i$ completely specify the geometry of the rod, we must also define discrete bending, and twisting strains to calculate a rod's elastic energy. The discrete bending strain is related to the curvature of the centerline, and is defined by the curvature binormal vector at each vertex:

$$(\kappa \mathbf{b})_i = \frac{2\mathbf{t}^i \times \mathbf{t}^{i-1}}{1 + \mathbf{t}^i \cdot \mathbf{t}^{i-1}} = \frac{2\mathbf{e}^i \times \mathbf{e}^{i-1}}{\|\mathbf{e}^i\|\|\mathbf{e}^{i-1}\| + \mathbf{e}^i \cdot \mathbf{e}^{i-1}}. \tag{2.1}$$

This vector is perpendicular to the plane that locally contains the rod (the osculating plane) and has magnitude $\kappa_i = \|(\kappa\mathbf{b})_i\| = 2\tan(\phi_i/2)$, where ϕ_i is the angle between successive tangents $\mathbf{t}^{i-1}$ and $\mathbf{t}^i$. This definition of curvature has some important structure-preserving properties [2], and is properly considered an integrated quantity, so that κ_i/l_i approximates the curvature $\kappa(s)$ of a continuous curve. Because a rod with anisotropic cross section may respond differently to bending in different planes, the curvature must be separated into components along the $\mathbf{d}_1$ and $\mathbf{d}_2$ axes. To do so, the vertex-based curvature is projected onto the average of the neighboring edge frames, so that

$$\boldsymbol{\kappa}_i = \begin{bmatrix} \kappa_{i,1} \\ \kappa_{i,2} \end{bmatrix} = \begin{bmatrix} (\kappa\mathbf{b})_i \cdot (\mathbf{d}_2^{i-1} + \mathbf{d}_2^i)/2 \\ -(\kappa\mathbf{b})_i \cdot (\mathbf{d}_1^{i-1} + \mathbf{d}_1^i)/2 \end{bmatrix}. \tag{2.2}$$

This projection includes a rotation by 90° about each tangent vector, so that $\boldsymbol{\kappa}_i/l_i$ approximates the curvature normal $\kappa\mathbf{n} = \mathbf{t}'(s)$ of a continuous rod instead of the curvature binormal $\kappa\mathbf{b} = \mathbf{t}(s) \times \mathbf{t}'(s)$.

The twist of the rod about its centerline can be quantified by comparing the directors on adjacent edges. Because these directors live in separate planes, its inconvenient to do this directly. Instead, we rely on the notion of parallel transport along a curve. Given two unit vectors $\mathbf{t}_1$ and $\mathbf{t}_2$, let $\mathbf{R}(\mathbf{t}_1, \mathbf{t}_2)$ be the unique rotation matrix that satisfies

$$\mathbf{R}\mathbf{t}_1 = \mathbf{t}_2, \tag{2.3}$$

$$\mathbf{R}(\mathbf{t}_1 \times \mathbf{t}_2) = \mathbf{t}_1 \times \mathbf{t}_2. \tag{2.4}$$

Intuitively, $\mathbf{R}$ rotates $\mathbf{t}_1$ into $\mathbf{t}_2$ about the axis $\mathbf{t}_1 \times \mathbf{t}_2$. The parallel transport of a director $\mathbf{d}^{i-1}$ from edge $\mathbf{e}^{i-1}$ to edge $\mathbf{e}^i$ is defined to be $P_{i-1}^i(\mathbf{d}^{i-1}) = \mathbf{R}(\mathbf{t}^{i-1}, \mathbf{t}^i)\mathbf{d}^{i-1}$. With this definition, the discrete twist τ_i is the angle between the director $\mathbf{d}_1^i$ and the transported director $P_{i-1}^i(\mathbf{d}^{i-1})$. Computationally, τ_i can be determined from the inner products

$$\cos(\tau_i) = \mathbf{d}_1^i \cdot \mathbf{R}(\mathbf{t}^{i-1}, \mathbf{t}^i)\mathbf{d}_1^{i-1} \tag{2.5}$$

$$\sin(\tau_i) = \mathbf{d}_2^i \cdot \mathbf{R}(\mathbf{t}^{i-1}, \mathbf{t}^i)\mathbf{d}_1^{i-1}. \tag{2.6}$$

Using the previous definition, parallel transport via the matrix $\mathbf{R}(\mathbf{t}^{i-1}, \mathbf{t}^i)$ is not defined for $\mathbf{t}^i = \pm\mathbf{t}^{i-1}$. When the two tangents are equal, its convenient to define $\mathbf{R}(\mathbf{t}, \mathbf{t})$ as the identity matrix, since $\lim_{\mathbf{t_1} \to \mathbf{t_2}} \mathbf{R}(\mathbf{t}_1, \mathbf{t}_2) = \mathbf{I}$. When the tangents are opposite, the corresponding limit does not exist, and parallel transport is truly ill-defined. This inability to handle full 180° kinks is a small drawback of DER, and also affects the curvature $\kappa_i = 2\tan(\phi_i/2)$, which become infinite for $\phi_i = \pi$.

2.2. Elastic Energy, Forces, Moments, and Hessians.. To measure the elastic energy of a rod, its necessary to define a resting configuration in which the rod has no stored elastic energy. Quantities in the resting configuration are denoted by an overbar, so that $\bar{l}^i$ represents the undeformed length of an edge. The full reference centerline $\{\bar{\mathbf{x}}_i\}$ and directors $\{\bar{\mathbf{d}}_1^i\}$ are not required; only the resting lengths $\bar{l}^i$, resting curvatures $(\bar{\kappa}_{i,1},\ \bar{\kappa}_{i,2})$, and resting twists $\bar{\tau}_i$ need to be retained. The rod's elastic modulus E and shear modulus G, along with the section properties A and $\mathbf{I}$, determine three distinct stiffness coefficients: a stretching stiffness $k^i = EA^i$, bending stiffness $\mathbf{B}_i = E\mathbf{I}_i$, twisting stiffness $\beta_i = GJ$. The elastic energies due to stretching, bending, and twisting are then

$$E_\epsilon = \frac{1}{2}\sum_i k^i (l^i - \bar{l}^i)^2/\bar{l}^i, \tag{2.7}$$

$$E_\kappa = \frac{1}{2}\sum_i (\boldsymbol{\kappa}_i - \bar{\boldsymbol{\kappa}}_i)^T \mathbf{B}_i (\boldsymbol{\kappa}_i - \bar{\boldsymbol{\kappa}}_i)\bar{l}_i, \tag{2.8}$$

$$E_\tau = \frac{1}{2}\sum_i \beta_i (\tau_i - \bar{\tau}_i)^2/\bar{l}_i, \tag{2.9}$$

and the total elastic energy is $E_{\text{el}} = E_\epsilon + E_\kappa + E_\tau$.

Having defined an energy function on the space of rod configurations, the elastic forces and twisting moments acting on the rod are taken to be the partial derivatives of the elastic energy. To calculate these derivatives, we consider two kinds of infinitesimal perturbations. In the first, the vertex $\mathbf{x}_i$ is translated an infinitesimal distance $\delta\mathbf{x}_i$. In order to remain perpendicular to the centerline, the neighboring directors $\mathbf{d}^i$ and $\mathbf{d}^{i-1}$ are parallel transported from the previous tangent vectors $\mathbf{t}^{i-1}$ and $\mathbf{t}^i$ to the infinitesimally perturbed tangent vectors. The resulting change in elastic energy defines the force acting on vertex $\mathbf{x}_i$, through

$$\mathbf{F}_i = \frac{\delta E_{\text{el}}}{\delta \mathbf{x}^i}. \tag{2.10}$$

In the second type of perturbation, the directors $\mathbf{d}_1^i$ and $\mathbf{d}_2^i$ are rotated by an infinitesimal angle $\delta\theta^i$ about the tangent $\mathbf{t}^i$, and the resulting change in energy defines the twisting moment

$$T^i = \frac{\delta E_{\text{el}}}{\delta \theta^i}. \tag{2.11}$$

These forces and moments can be calculated analytically, and the corresponding expressions are given in [2], [1], and [6]

For implicit time integration or quasi-static problems, its also helpful to have analytical expressions for the second derivatives $\delta^2 E_{\text{el}}/\delta\mathbf{x}_i\delta\mathbf{x}_j$, $\delta^2 E_{\text{el}}/\delta\mathbf{x}_i\delta\theta_j$, and $\delta^2 E_{\text{el}}/\delta\mathbf{x}_i\delta\theta_j$. These can also be derived analytically, though the resulting expressions are frighteningly complex. If the vertex and edge degrees of freedom are concatenated into a vector $q = [\mathbf{x}_0,\ \theta^0,\ \mathbf{x}_1,\ \theta^1,\ ...,\ \theta^n,\ \mathbf{x}_{n+1}]$, the resulting Hessian $\delta^2 E/\delta q^2$ is a symmetric matrix with bandwidth 10.

2.3. Reduced coordinates. In the previous section, a single rotational degree of freedom θ^i was assigned to each edge. This reflects the fact that while each director $\mathbf{d}_1^i$ has three components, it is constrained to be unit length and orthogonal to $\mathbf{t}^i$. Because of these constraints, listing the vertices $\{\mathbf{x}_i\}$ and directors $\{\mathbf{d}_1^i\}$ is a poor way to parametrize the space of all rod configurations, since many such combinations are not admissible rods.

To be compatible with existing optimization and ODE integration packages, its necessary to parametrize the space of configurations so that there is a one-to-one correspondence between coordinates and degrees of freedom. A convenient way to do this is to fix a reference configuration (distinct from the resting configuration) with vertices $\{\mathbf{x}_i\}$ and directors $\{\mathbf{d}_1^i\}$, and relate all other configuration to this reference by a set of translational displacements $\{\Delta\mathbf{x}_i\}$ and rotational displacements $\{\Delta\theta^i\}$. The corresponding configuration is obtained by calculating the new centerline $\{\mathbf{x}_i + \Delta\mathbf{x}_i\}$, parallel transporting the directors from the centerline to the new, and then rotating each director by an angle $\Delta\theta^i$.

The major disadvantage to this parametrization is that it can not handle any configuration which reverses the orientation of an edge, leading to ill-defined parallel transport. In practice, this is remedied by frequently updating the reference configuration, so that the old and new centerlines remain close in orientation.

2.4. Boundary Conditions. The specification of boundary conditions for Discrete Elastic rods is still an active research area. Simple conditions like pinned or clamped ends can be implemented by fixing one or more degrees of freedom on either end of the rod. In [2], Bergou et al. establish methods for one end of a rod to a rigid body, which is necessary for animating realistic hair. More recently, Lestringant and Kochmann have proposed a new method for enforcing rigid connections between rods in a lattice configuration [7], which relies on a collection of phantom rod edges. For simplicity, this project considers only free and fully clamped BCs.

3. Implementation: DiscreteElasticRods.jl. An effective DER implementation must be able to handle a wide variety of simulation scenarios. Any rod simulation is defined at minimum by a collection of rods, each one subject to a boundary condition at either end. Some boundary conditions involve only a single rod, as in a clamped or free end, while others involve multiple rods, as in a four-way junction. Each rod has a definitive starting vertex $\mathbf{x}_0$ and ending vertex $\mathbf{x}_N$, which are determined by the direction of the parametrization.

These conditions are precisely described by a directed graph, where each edge represents a rod and each node represents a boundary condition. When combined with the details of each boundary condition, the resting strains in each rod, and an initial condition, this graph completely specifies a particular rod problem. Ideally, this could graph could be implemented with Julia's LightGraphs.jl; however, the LightGraphs main interface forbids the duplication of edges, which is a perfectly valid scenario for a physical rod system. An easy alternative is simply to create a list of rods, and store with each boundary condition the index of the rod (and start/end orientation) to which it should be applied

Ideally, a DER implementation can function without demanding any more information than this specification graph from the user. All of the degrees of freedom in the problem can be determined from the graph, and assembled into a single vector q. An elastic energy function $E(q)$ on these degrees of freedom can then built automatically, and then minimized to find an equilibrium configuration of the structure.

To make this tractable, DiscreteElasticRods.jl divides this task into two parts. The first is a "specification layer". This code accepts a graph representing a rod problem, and returns the number of degrees of freedom N and a function which maps an array N real numbers to a set of coordinates $\left\{\Delta\mathbf{x}_i, \Delta\theta^i\right\}$ for each rod. The returned function is parametrized on the type of the input array, so that dual-number based AD propagates seamlessly through this layer. The second half of the code is the "Energy Layer", which accepts the coordinates for each rod and calculates the total elastic energy of the system. This layer is not problem-specific, and can be optimized as a standalone function without reference to any particular rod problem. With these two layers in place, a gradient-based minimization of the elastic energy can be performed by automatically differentiating through the composition of the specification and energy layers, directly giving the gradient and Hessian (or Hessian vector products) of the elastic energy with respect to each coordinate degree of freedom.

Although the current implementation of the specification layer currently accepts only clamped and free boundary conditions, it has been designed to accommodate considerably more boundary conditions. A user wishing to add a new BC needs only to specify the number of associated free parameters, and a function that maps these parameters to the first (or last) three coordinates of the rod ($\mathbf{x}_0,\ \theta^1,\ \mathbf{x}_1$). Because this portion of the code is largely logistical, and has little performance impact, the remainder of this work concentrates on the energy layer.

4. Performance Analysis of the Energy Layer.

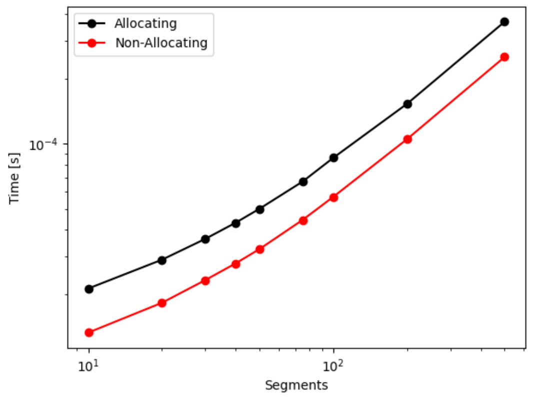
The energy layer has a simple job: to accept a reference configuration and a set of displacements for one rod, and return the elastic energy, its gradient, or its Hessian. This part of the code is not problem-specific, so its performance varies only with the number of segments in the rod. Because the goal of this project is to avoid the specification of any analytical Jacobians or Hessians, this section focuses entirely on the performance of Julia's AD ecosystem.

Here we consider three different types of automatic differentiation: forward mode via dual numbers, implemented by ForwardDiff.jl[10]; tracing reverse-mode, implemented by ReverseDiff.jl; and source-to-source, as implemented by Zygote.jl [4]. Because Zygote.jl does not support mutation, the rod kinematics and elastic energy function are written in a non-mutating style, which has a profound effect on the relative performance of the three packages. The elastic energy function also has no control-flow, so that tracing Reverse-Mode AD can trace the execution of the function just once and then compile the reverse pass for all possible inputs.

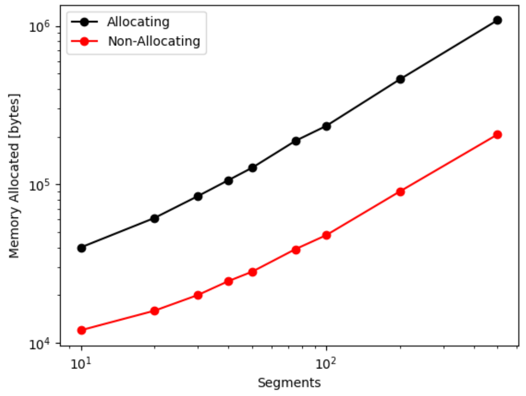
Task zero for the energy layer is simply to evaluate the elastic energy. Although this involves no AD, and is a small task compared to taking gradients or Hessians, fast function evaluations are still a plus when using minimization algorithms that rely on a line search for robustness. Figure 2 examines the evaluation time of an allocating and (almost) non-allocating implementation of the rod kinematics and elastic energy. As expected, both have linear complexity in the length of the rod, with the non-allocating version outperforming the allocating version. Here the difference is roughly a factor of two in evaluation time.

The first task for each AD system is evaluating the gradient of the elastic energy. The time spent calculating these gradients for rods with between 10 and 500 segments is shown in Figure 3 for all three packages. ReverseDiff.jl outperforms both of the other packages across the entire size range, although it performance is only slightly above Zygote.jl for $N_s \geq 100$. ForwardDiff.jl matches or outperforms Zygote.jl for $Ns \leq 30$, but beyond that it is by far the least effective method. This is not surprising, since forward-mode has a double handicap in this task. The evaluation of gradients by forward-mode requires the use of a different dual component for each element of the gradient, as compared to the single pullback used in reverse mode. On top of this, ForwardDiff.jl is most effective when used in non-allocating functions. The use of a non-mutating coding style in the energy function precludes the use of cacheing, causing ForwardDiff.jl to allocate significantly more memory than either reverse mode implementation.

For a single rod, the Hessian of the elastic energy is a banded matrix, so that entire Hessian can be explicitly evaluated and inverted in $\mathcal{O}(N_s)$ time. Because DiscreteElasticRods.jl is aimed more general problems, where interactions between rods spoil the banded structure of this matrix, the focus here is not on Hessian construction but rather on Hessian-Vector products (HVPs). To that end, the evaluation times for all five varieties of the out-of-place HVPs provided by SparseDiffTools.jl are shown in Figure 4 across a range of rod sizes. For all but the smallest problem size, the fully



(a)



(b)

Fig. 2: Evaluation time (a) and memory allocation (b) for allocating and less-allocating implementations of the DER kinematics and elastic energy functions.

automatic “AutoBack” routine consisting of a forward mode JVP through Zygote’s reverse-mode gradient is the fastest option.

Even without a banded structure, the locality of the physics on each rod leads to extremely sparse Hessians for large lattices of elastic rods. For Hessian-based minimization, these must matrices be inverted iteratively using a Krylov subspace method. Thus the last task evaluated here is evaluating an inverse-Hessian vector product, using the iterative linear solvers implemented in IterativeSolvers.jl. Because the Hessian is not necessarily positive definite for some rod configurations, its not possible to use a Conjugate Gradient method. However, we can still take advantage of symmetry by using the MinRes algorithm, which has a memory advantage over the more general GMRes algorithm. Figure 5 compares the performance of MinRes implemented over two different HVP operators: numerical with forward mode (NumAuto), and

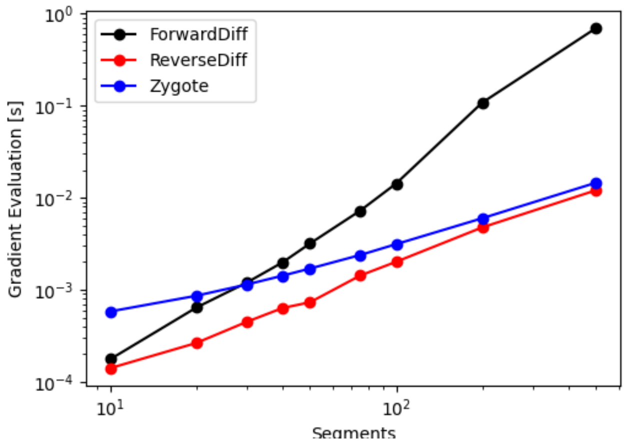


Fig. 3: Time to evaluate gradients of elastic energy using three different AD packages.

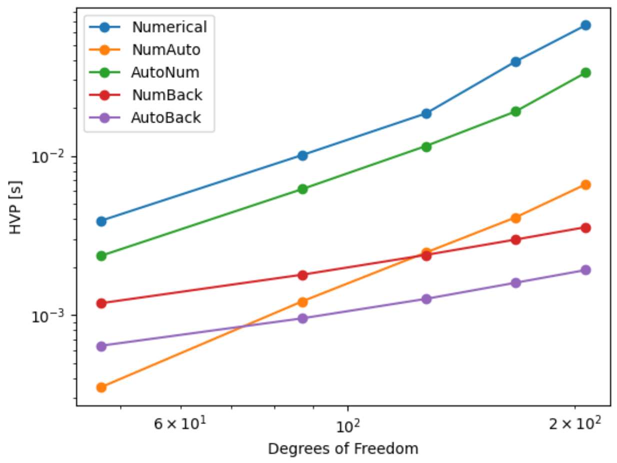


Fig. 4: Time to evaluate Hessian-vector products using the five AD strategies implemented in SparseDiffTools.jl.

forward-over-reverse mode (AutoBack). The results mirror those of the HVP timings, indicating that forward-over-reverse mode AD is the faster of the two options for all but the smallest problem size.

(As a final note, in order to use the forward-over-reverse mode HVP provided by SparseDiffTools.jl within IterativeSolvers, its necessary to wrap the forward-over-reverse mode HVP in an operator structure. It would be awesome to see this wrapper implemented in a future version of SparseDiffTools.jl, as it already is for the num-auto HVP.)

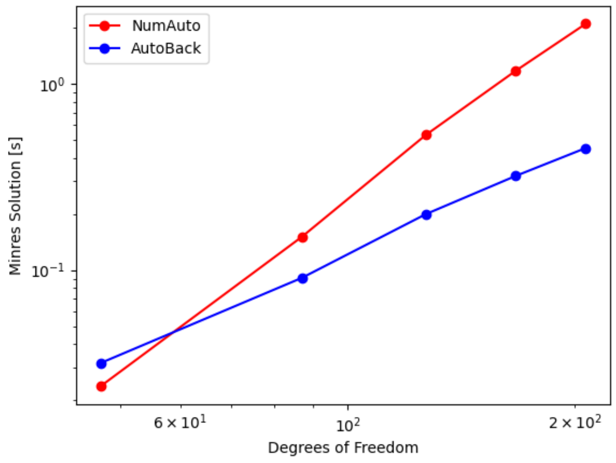


Fig. 5: Time to solve linear system involving the Hessian using MinRes with automatic HVPs. Each solve was provided the same right hand side and initial guess for the solution.

5. Future Directions for DER in Julia.

To wrap up the discussion, this section touches on three immediate extensions of the implementation discussed above.

5.1. Analytical Jacobians.

While this report demonstrated forward-over-reverse mode automatic differentiation to calculate Hessian-Vector products, this could easily be improved upon in the current Julia AD ecosystem. If the gradient of elastic energy is implemented analytically, its no longer necessary to use a non-mutating codebase to stay compatible with Zygote.jl. With non-allocating analytical gradients, Hessian-vector products could be evaluated using forward-mode AD with single-component dual numbers and zero allocations. By avoiding the analytical Hessian but still allowing for exact non-allocating HVPs, this strategy represents an excellent compromise between speed and code complexity.

5.2. Dynamic Problems.

Although this report emphasizes quasi-static equilibrium problems, DER is also a dynamic model. After assigning masses $m_i = \rho A_i \bar{l}_i$ to each vertex and a (mass) moment of inertia $I^i = \rho \mathbf{l}^i J^i$ to each edge, the equations of motion for a rod take the form

$$m_i \ddot{\mathbf{x}}_i = \mathbf{F}_i(\{\mathbf{x}_i\}, \{\theta^i\}) + \mathbf{F}_i^{\text{external}},$$
$$I^i \ddot{\theta}^i = T^i(\{\mathbf{x}_i\}, \{\theta^i\}) + T^i_{\text{external}}.$$

In a typical rod problem, the bending, stretching, and twisting degrees of freedom evolve on three separate and increasingly small timescales. The inertias I^i are proportional to the fourth power of rod thickness, and are usually vanishingly small. Consequently, the fastest twisting dynamics can be eliminated by using a quasi-static approximation in which $I^i = 0$. This leads to a first order DAE for $\mathbf{x}_i$, θ^i, and the velocities $\mathbf{v}_i$:

$$\begin{aligned} \dot{\mathbf{x}}_i &= \mathbf{v}_i, \\ m_i \dot{\mathbf{v}}_i &= \mathbf{F}_i(\{\mathbf{x}_i\}, \{\theta^i\}) + \mathbf{F}_i^{\text{external}}, \\ 0 &= T^i(\{\mathbf{x}_i\}, \{\theta^i\}) + T^i_{\text{external}}. \end{aligned}$$

Because the stretching dynamics are still much faster than the bending dynamics, this system is best integrated implicitly. All of the strategies developed here for evaluating energy gradients and Hessians apply equally well to the nonlinear solves used in this implicit integration.

5.3. Parallelism. Because rods are one-dimensional structures, rod problems tend to have a low memory requirement compared to 2D or 3D PDE simulations. For scale, if strands of hair are discretized using 500 discrete segments each, then 250 MB of memory is sufficient to store the centerline $\{\mathbf{x}_i\}$ and directors $\{\mathbf{d}_1^i\}$ of roughly 10,000 strands of hair. This makes rod simulations well-suited for shared memory parallelism, since even extremely large problems can fit comfortably within a single HPC node. In the specification-solution setup described earlier, gradients or hessian-vector products can be calculated.

All code used to generate the results in this report is available on GitHub at github.com/jamesgabbard/DiscreteElasticRods.jl.

REFERENCES

[1] M. Bergou, B. Audoly, E. Vouga, M. Wardetzky, and E. Grinspun, *Discrete viscous threads*, ACM Transactions on Graphics (TOG), 29 (2010), pp. 1–10.
[2] M. Bergou, M. Wardetzky, S. Robinson, B. Audoly, and E. Grinspun, *Discrete elastic rods*, in ACM SIGGRAPH 2008 papers, 2008, pp. 1–12.
[3] J. W. Boley, W. M. van Rees, C. Lissandrello, M. N. Horenstein, R. L. Truby, A. Kotikian, J. A. Lewis, and L. Mahadevan, *Shape-shifting structured lattices via multimaterial 4d printing*, Proceedings of the National Academy of Sciences, 116 (2019), pp. 20856–20862.
[4] M. Innes, A. Edelman, K. Fischer, C. Rackauckas, E. Saba, V. B. Shah, and W. Tebbutt, *A differentiable programming system to bridge machine learning and scientific computing*, CoRR, abs/1907.07587, (2019).
[5] M. K. Jawed, F. Da, J. Joo, E. Grinspun, and P. M. Reis, *Coiling of elastic rods on rigid substrates*, Proceedings of the National Academy of Sciences, (2014), p. 201409118.
[6] C. Lestringant, B. Audoly, and D. M. Kochmann, *A discrete, geometrically exact method for simulating nonlinear, elastic and inelastic beams*, Computer Methods in Applied Mechanics and Engineering, 361 (2020), p. 112741.
[7] C. Lestringant and D. M. Kochmann, *Modeling of flexible beam networks and morphing structures by geometrically exact discrete beams*, Journal of Applied Mechanics, 87 (2020).
[8] F. Ling, H. Guo, and E. Kanso, *Instability-driven oscillations of elastic microfilaments*, Journal of The Royal Society Interface, 15 (2018), p. 20180594.
[9] A. Novelia, *Discrete Elastic Rods for Simulating Soft Robot Limbs*, PhD thesis, UC Berkeley, 2018.
[10] J. Revels, M. Lubin, and T. Papamarkou, *Forward-mode automatic differentiation in julia, 2016*, URL https://arxiv. org/abs/1607.07892.