Dynamic Simulation

TODOS

- What is dynamic simulation?
- What is the standard approach?
- What are the challenges?
- Enter constraints!

Simulating effects such as incompressibility, inextensibility and joints between articulated rigid bodies in elasticity-based simulations can be achieved by using high stiffness values. High stiffness values lead to large forces which in turn cause numerical issues in the solver.

We demonstrate these issues based on the example of maintaining a desired distance between two points using a stiff spring [15]. Let $\boldsymbol{x_1}, \boldsymbol{x_2}$ be the positions, $\boldsymbol{v_1}, \boldsymbol{v_2}$ the velocities and $\boldsymbol{a_1}, \boldsymbol{a_2}$ be the accelerations of the two particles. Let $\bar{l}$ be the rest length and $l = \|\boldsymbol{x_1} - \boldsymbol{x_2}\|$ be the current length of the spring with stiffness k. It can be shown that the force that the spring applies at each particle is equal to $\boldsymbol{f_1} = -\boldsymbol{f_2} = \lambda \boldsymbol{u}$, where $\boldsymbol{u} = (\boldsymbol{x_1} - \boldsymbol{x_2})/l$ and $\lambda = -\frac{\delta V}{\delta l} = k(\bar{l} - l)$.

Once the forces, accelerations, velocities and positions are combined into vectors $\boldsymbol{f}, \boldsymbol{a}, \boldsymbol{v}, \boldsymbol{x}$, respectively, the motions of the system can be modeled via Newton's Ordinary Differential Equation (ODE) $\boldsymbol{f} = \boldsymbol{M}\boldsymbol{a}$, where $\boldsymbol{M}$ is a $n_d \times n_d$ diagonal matrix and n_d is the total number of independent degrees of freedom for the particles.

This system can be integrated via the symplectic Euler method as follows:

$$\boldsymbol{v_{n+1}} = \boldsymbol{v_n} + h\boldsymbol{a_n}$$
$$\boldsymbol{x_{n+1}} = \boldsymbol{x_n} + h\boldsymbol{v_{n+1}}$$

As the stiffness k of the spring increases, so does the magnitude of the acceleration $\boldsymbol{a}$. Consequently, the integration diverges unless the timestep is prohibitively small. The stability issues are often addressed by switching to an implicit integration scheme, such as the backward Euler method [2]. Replacing current accelerations with future accelerations requires the solution of the following linear system of equations (LSE):

$$(\boldsymbol{M} - h^2\boldsymbol{K})\boldsymbol{v_{n+1}} = \boldsymbol{p} + h\boldsymbol{f}$$

where $\boldsymbol{p} = \boldsymbol{M}\boldsymbol{v_n}$ is the momentum, and $\boldsymbol{K} = \frac{\delta \boldsymbol{f}}{\delta \boldsymbol{x}}$ is the stiffness matrix. Note that $\boldsymbol{K}$ is typically non-singular since elastic forces are invariant under rigid body transforms. When using large stiffness k for springs, the entries of $\boldsymbol{K}$ are large (due to large restorative forces for stiff springs) and dominate the entries of the system matrix $\boldsymbol{H} = \boldsymbol{M} - h^2\boldsymbol{K}$. In these cases, $\boldsymbol{H}$ will be almost non-singular as well, leading to numerical issues and poor convergence for many solvers. Additionally, implicit integration introduces noticable numerical damping [13].

This system results from performing the implicit integration and solving the non-linear system via linearization using the Taylor expansion. Positions can be expressed in terms of velocities and eliminated from the system.

Penalty Forces

In the example above, the energy was derived from Hooke's Law for springs. However, it is also possible to derive energies from geometric displacement functions $\boldsymbol{\phi(x)}$ which vanish in the rest configuration. From the displacement functions, quadratic potential energies of the form $U(\boldsymbol{x}) = \Sigma_i (k/2)\boldsymbol{\phi^2(x)}$, where k is a positive stiffness parameter, are constructed [14]. The potential energy $U(\boldsymbol{x})$ is zero if the displacement function is satisfied, and greater than zero otherwise. The resulting forces are called penalty forces. Using the geometric displacement function $\boldsymbol{\phi}_{\text{spring}}(\boldsymbol{x}) = (\|\boldsymbol{x_i} - \boldsymbol{x_j}\|) - l$ with k_{spring} recovers the behavior of a spring with stiffness k_{spring}. Its displacement function $\boldsymbol{\phi}_{\text{spring}}(\boldsymbol{x})$ is satisfied when the distance of the particles $\boldsymbol{x_i}, \boldsymbol{x_j}$ is equal to a desired rest length l. By constructing different geometric displacement functions, various properties such as the bending angle between triangles and in-plane shearing of triangles can be controlled via the corresponding quadratic energy potentials [2]. Geometric displacement functions with the desired effect are often intuitive and simple to define. However, as the corresponding energy potentials are not physically derived, choosing stiffness parameters that correspond to measurable physical properties of the simulated

material and orchestrating multiple constraints becomes challenging [13, 10]. Additionally, the generated penalty forces do not converge in the limit of infinite stiffess, leading to oscillations unless the timestep is reduced significantly [12].

Maybe explain the challenges with penalty forces a bit better! Also read [14, 10, 12]. I just skimmed over [14] for now, but want to make sure that I am citing this correctly. The term penalty forces is not used in the paper, I am just following the trail from [13]. [10] is a review that might be intersting to read. [12] would be really interesting to read for once, just to understand why strong penalty forces oscillate. Is this a general problem with penalty forces, or is it an issue with the solver?

Mass Modification

The motion of a particle can be influenced by modifying the inverse mass matrix $\boldsymbol{M^{-1}}$ of the system. For a single particle $\boldsymbol{x_i}$, it is $\ddot{\boldsymbol{x}}_{\boldsymbol{i}} = \boldsymbol{M_i^{-1}} \boldsymbol{f}$, where $\boldsymbol{M_i^{-1}}$ is the inverse mass matrix for particle i. If, for example, the first diagonal entry of $\boldsymbol{M_i^{-1}}$ is zero, no acceleration in the x-direction is possible. It is possible to construct modified inverse mass matrices $\boldsymbol{W}$ such that the accelerations in the three axes of an arbitraty orthogonal coordinate system can be restricted. The modified inverse mass matrices $\boldsymbol{W}$ can be used in the LSE that results from the implicit integration above. By adding simple velocity and position terms to the system equations the magnitude of the change in velocity in each direction and even the exact position of each constrained particle can be controlled. This approach is called mass modification [2]. In [2], the authors use mass modification to model collision constraints between objects and cloth and other user defined constraints.

Since the velocity and position of each constrained particle is controlled via a single velocity and position term, multiple constraints that affect the same particle have to be handled together. This can lead to constraints which affect arbitrarily many particles. For that reason, self-collisions of cloth are not handled via mass modification in [2]. Instead, penalty forces are used. Additionally, accurately constraining particle positions is only possible for particles whose velocity is constrained as well. The resulting system is unbanded, but sparse, and is solved using a modified version of the conjugate gradient (CG) method.

Constraint-based Dynamics

Hard Constraints

The problem of maintaining hard distance constraints between particles can be formulated as a Differential Algebraic Equation (DAE) [16, 1]. In this framework, the standard ODE $\boldsymbol{f} = \boldsymbol{Ma}$ is handled together with algebraic equations that model the constraints on the system. Distance constraints are typically implemented using holonomic constraints of the form $\boldsymbol{\phi(x)} = 0$. Note that the distance constraint $\boldsymbol{\phi}(x)$ is formulated in terms of the particle positions, whereas the ODE works on accelerations or velocities. Consequently, the constraints need to be differentiated once or twice so that they can be combined with the ODE in terms of velocties or accelerations, respectively. In xPBD, we go the other way! The ODE is tanslated so that it is in terms of positions, so that it can be handled together with the constraints. Is there a reason nobody bothered to do this before? What are the challenges here? Is this exactly what xPBD is? Is there a way to view the simplifications made in terms of the other frameworks?. Using $\boldsymbol{J} = \frac{\delta \boldsymbol{\phi}}{\delta \boldsymbol{x}}$, where $\boldsymbol{J}$ is a $n_c \times n_d$ matrix and n_c is the number of scalar constraints, this leads to the following constraint formulations:

$$\boldsymbol{Jv} = 0$$
$$\boldsymbol{Ja} = \boldsymbol{c(v)}$$

for some $\boldsymbol{c(v)}$. If you check [16], see that $c(v)$ also depends on the positions q. That should be indicated! Additionally, constraint forces are required in order to link the algebraic constraint equations with the ODE describing the motion of the system. It can be shown that the constraint forces $\boldsymbol{f_c}$ applied to the particles have to be in the following form in order to avoid adding linear and angular momentum to the system [1]:

$$\boldsymbol{f_c} = \boldsymbol{J^T \lambda}$$

where the λ are the Lagrange multipliers of the constraints. With external forces $\boldsymbol{f_e}$, the DAE can now be expressed as follows [16]:

$$\begin{pmatrix} \boldsymbol{M} & -\boldsymbol{J}^{\boldsymbol{T}} \\ \boldsymbol{J} & 0 \end{pmatrix} \begin{pmatrix} \boldsymbol{a} \\ \boldsymbol{\lambda} \end{pmatrix} = \begin{pmatrix} \boldsymbol{f_e} \\ \boldsymbol{c(v)} \end{pmatrix}$$

Note that the lower block-row of the system drives towards accelerations that satisfy the constraints imposed by $\boldsymbol{\phi(x)}$ (or, striclty speaking, the differentiations thereof) exactly. This is indicated by the lower-right zero block in the system matrix in either formulation. Thus, the system does not have a solution if constraints are contradictory. Aren't $\dot{q} = v$ and $\dot{v} = a$ also part of the differential equation? Because $c(v)$ and f_e also depend on q!

In [16], the DAE is approached by eliminating the λ from the system entirely and constructing an ODE in terms of positions and velocities. In [15], the authors suggest applying implicit integration schemes to the system directly by constructing the following Karush-Kuhn-Tucker (KKT) equation system:

$$\begin{pmatrix} \boldsymbol{M} & -\boldsymbol{J}^{\boldsymbol{T}} \\ \boldsymbol{J} & 0 \end{pmatrix} \begin{pmatrix} \boldsymbol{v_{n+1}} \\ \boldsymbol{\mu_{n+1}} \end{pmatrix} = \begin{pmatrix} \boldsymbol{p} + h\boldsymbol{f_e} \\ 0 \end{pmatrix}$$

Here, the external forces $\boldsymbol{f_e}$ and the constraint gradients $\boldsymbol{J}$ are considered constant across the timestep and $\boldsymbol{J(x_{n+1})}$ is not approximated using the Taylor expansion like it is in [2]. If internal forces are taken into account, the upper-left matrix $\boldsymbol{M}$ is replaced by the matrix $\boldsymbol{H}$ from above.

Reverse-engineering how the authors arrived at this system is quite enlightening. Start out from the equations of motion [16]

$$\boldsymbol{\dot{v} = M^{-1}(f - J^T)\lambda}$$

and perform implicit integration:

$$\begin{aligned} \boldsymbol{v_{n+1}} &= \boldsymbol{v_n} + h\boldsymbol{M^{-1}}(\boldsymbol{f_e(x_{n+1})} - \boldsymbol{J^T(x_{n+1})\lambda(x_{n+1})}) \\ \boldsymbol{Mv_{n+1}} &= \boldsymbol{p} + h\boldsymbol{f_e(x_{n+1})} - h\boldsymbol{J^T(x_{n+1})\lambda(x_{n+1})} \\ \boldsymbol{Mv_{n+1}} + h\boldsymbol{J^T(x_{n+1})\lambda(x_{n+1})} &= \boldsymbol{p} + h\boldsymbol{f_e(x_{n+1})} \\ \boldsymbol{Mv_{n+1}} + \boldsymbol{J^T(x_{n+1})\mu(x_{n+1})} &= \boldsymbol{p} + h\boldsymbol{f_e(x_{n+1})} \end{aligned}$$

If we assume that $\boldsymbol{f_e}$ and the constraint gradients $\boldsymbol{J}$ are constant across the time step, we arrive at the formulation from the paper. For the external forces, which are usually only comprised of gravitational forces, this is not a big deal. For the constraint gradients, I am not sure what the ramifications are. In [2], the Taylor expansion is performed which requires the compution of second derivatives over the constraint functions. This is not happening here at all! Is this what authors mean when they say that the constraints are effectively linearized during one solve, e.g. second page of [9]? Technically, speaking, even if the Taylor expansion is performed, the constraints are linearized, if I understand correctly.

Note that the system matrix is sparse, which can be exploited by sparse-matrix solvers in order to solve the system efficiently [1]. Alternatively, the Schur complement can be constructed since the mass matrix in the upper left block is invertible. This leads to a smaller, albeit less sparse system [15]:

$$\boldsymbol{JM^{-1}J^T\mu = -JM^{-1}(p + hf_e)}$$

If the constraints are not redundant, $\boldsymbol{JM^{-1}J^T}$ is non-singular and symmetric positive definite [1], which are desirable properties for many solvers. According to [13], the common approaches for linearizing the constraint forces and stabilizing the constraints $\boldsymbol{\phi(x)} = 0$ are notoriously unstable. Additionally, instabilities in the traverse direction of the constraints occur when the tensile force with respect to particle masses is large when using hard constraints [15].

Compliant Constraints

By combining ideas from hard constraints and penalty forces, it is possible to formulate the system matrix for hard constraints such that constraints do not have to be enforced exactly. In this approach, called compliant constraints, the constraints are combined with the the ODE $\boldsymbol{f = Ma}$ in a way that allows relaxation of constraints in a physically meaningful manner [13]. The key insight is that constraints of the form $\boldsymbol{\phi(x)}$ are the physical limit of strong potential forces of the form $\frac{k}{2}\boldsymbol{\phi^2(x)}$ with high stiffness values k. However, using large, but finite, stiffness values has adverse

affects on the numerical properties of the system matrix. Thus, the equations of motion are rewritten in terms of the inverse stiffness. The potential energy for the constraint $\boldsymbol{\phi}$ is then defined as:

$$U(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{\phi^T}(\boldsymbol{x})\alpha^{-1}\boldsymbol{\phi}(\boldsymbol{x})$$

where α is a symmetric, positive definite matrix of dimension $n_c \times n_c$. The correspondence to the penalty terms above is the case where α^{-1} is a diagonal matrix with diagonal entries $\frac{1}{k_i}$ for the stiffness k_i of constraint $\boldsymbol{\phi(x)}$. The resulting forces $\boldsymbol{f_c} = \delta U/\delta \boldsymbol{x} = -J^T\alpha^{-1}\boldsymbol{\phi}$. In order to replace the large parameters α^{-1} with the small α in the equations of motion, artificial variables $\lambda = -\alpha^{-1}\boldsymbol{\phi}$ are introduced, yielding $\boldsymbol{f_c} = J^T\lambda$.

This leads to the following DAE:

$$\begin{aligned}\dot{\boldsymbol{x}} &= \boldsymbol{v}\\ \boldsymbol{M}\dot{\boldsymbol{v}} &= \boldsymbol{f_e} + \boldsymbol{J^T\lambda}\\ \alpha\lambda(\boldsymbol{x},t) &= -\boldsymbol{\phi}(\boldsymbol{x},t)\end{aligned}$$

Note, that in the limit of infinite stiffness, the formulation from hard constraints is recovered. By performing backwards differentiation and assuming that the constraint gradients are constant across the timestep, it is:

$$\boldsymbol{\alpha\lambda_{n+1}} = \boldsymbol{C}\frac{\boldsymbol{\mu_{n+1}}}{h} = -\boldsymbol{\phi_{n+1}} \approx -\boldsymbol{\phi} - h\boldsymbol{J}\boldsymbol{v_{n+1}}$$

leading to the following LSE [15]:

$$\begin{pmatrix}\boldsymbol{M} & -\boldsymbol{J^T}\\ \boldsymbol{J} & \frac{1}{h^2}\boldsymbol{\alpha}\end{pmatrix}\begin{pmatrix}\boldsymbol{v_{n+1}}\\ \boldsymbol{\mu_{n+1}}\end{pmatrix} = \begin{pmatrix}\boldsymbol{p} + h\boldsymbol{f_e}\\ -\frac{1}{h}\boldsymbol{\phi}\end{pmatrix}$$

Regarding the backwards differentiation above, this does not perform the Taylor approximation again. It should be something like:

$$\begin{aligned}\phi_+ \approx \phi + h\dot{\phi_+} &= \phi + hJ_+v_+\\ &\approx \phi + h(J + h\dot{J})v_+\\ &= \phi + h(J + h\frac{\delta J}{\delta x}\frac{\delta x}{\delta t})v_+\\ &= \phi + h(J + h\frac{\delta J}{\delta x}v)v_+\end{aligned}$$

Now, we need second derivatives of the constraints. This can be seen in [2] and is also mentioned in [13].

This formulation comes with a couple of advantages. Firstly, relaxing the constraints by keeping a finite but large penalty parameter helps counteracting numerical problems in the presence of over defined or degenerate constraints. Secondly, in comparison to the system from hard constraints, introducing α in the lower right block of the system matrix makes the system matrix strongly positive definite, which is beneficial for many solvers. Lastly, in comparison to penalty forces, entries of large magnitudes in the system matrix due to high stiffness terms are exchanged for small entries in terms of inverse stiffness, which improves the condition number of the matrix.

All these concepts from numerics are a bit unclear to me. I might have to go back to some textbook and do some reading to improve my understanding. I might have to go back to some textbook and do some reading to improve my understanding. Not sure the last part is entirely true.

In [13], a solver based on symplectic Euler which does not require second derivatives is derived. I do not understand some of the estimations made in that derivation. In particular the mean of a function f over and interval (a, b) is defined as $\frac{1}{b-a}\int_a^b f(x)dx$, so what they are saying does not make a lot of sense.

Projective Dynamics

In the approaches to physical simulations via implicit time integration that we have encountered so far, a new linear system needs to be solved at every timestep. For large simulations, tackling a new, different linear system of equations every time step can quickly become prohibitively expensive. At each timestep, new factorizations for the system matrix need to be computed. In PBD, this issue is dealt with by using an iterative solver. In Projective Dynamics (PD), a different approach is used. Energy potentials are restricted to a specific structure which allow for efficient implicit time integration via alternating steps of local and global optimization. The local optimization steps are comprised of per-constraint projections of particle positions onto constraint manifolds. The global optimization step combines the results from the individual local projection steps while taking into consideration global effects including inertia and external forces. This is achieved by solving a linear system of equations whose system matrix is constant across timesteps. Since the local steps can be carried out in parallel and the factorization for the system matrix of the global step can be precomputed and reused, physical simulations that are restricted to energy potentials from the PD framework can be solved efficienty and robustly.

Energy Potentials

Let the positions of m particles in a mesh be stored in a matrix $\boldsymbol{q} \in \mathbb{R}^{m\times 3}$ with deformation gradient $\boldsymbol{F} \coloneqq \boldsymbol{F}(\boldsymbol{q}) \in \mathbb{R}^{3\times 3}$. Then, energy potentials of the general form $\psi(\boldsymbol{E}(\boldsymbol{F}))$, where $\boldsymbol{E}(\boldsymbol{F})$ is a strain measure that depends on the deformation gradient of a discrete element, are frequently used in nonlinear continuum mechanics. If $\boldsymbol{E}$ is Green's strain measure $\boldsymbol{E}_{\text{Green}}$ defined by

$$\boldsymbol{E}_{\text{Green}}(\boldsymbol{F}) = \frac{1}{2}(\boldsymbol{F}^{\boldsymbol{T}}\boldsymbol{F} - \boldsymbol{I})$$

then $\boldsymbol{E}_{\text{Green}}(\boldsymbol{F}) = 0$ is equivalent to $\boldsymbol{F}^{\boldsymbol{T}}\boldsymbol{F} = \boldsymbol{I}$. Thus, $\boldsymbol{E}_{\text{Green}}(\boldsymbol{F}) = 0$ defines a constraint manifold that accepts deformations whose deformation gradients $\boldsymbol{F}$ are rotation matrices. These deformations are exactly the rigid-body transforms, i.e. transforms that alter the body's position and orientation but keep the body's volume undeformed. Assuming that $\psi(\boldsymbol{0}) = \rho$ for some $\rho \in \mathbb{R}$ and that ψ reaches its minimum at the undeformed configuration, then

$$d(\boldsymbol{E}_{\text{Green}}(\boldsymbol{F})) = \psi(\boldsymbol{E}_{\text{Green}}(\boldsymbol{F})) - \rho$$

can be considered a distance measure of how far the configuration is from the constraint manifold defined by the undeformed configurations.

The energy potentials in PD are designed to fit into this framework: Energy potentials are defined by a constraint manifold $\boldsymbol{C}$ – which can be different from $\boldsymbol{E}_{\text{Green}}(\boldsymbol{F}) = 0$ – and a distance measure d of the body's current configuration to that constraint manifold. Formally, this leads to energy potentials which of the following form:

$$\psi(\boldsymbol{q}) = \min_{\boldsymbol{p}} d(\boldsymbol{q}, \boldsymbol{p}) + \delta_{\boldsymbol{C}}(\boldsymbol{p}).$$

Here, $\boldsymbol{p} \in \mathbb{R}^{r\times 3}, r \in \mathbb{N}$ are auxiliary projection variables and $\delta_{\boldsymbol{C}}(\boldsymbol{p})$ is an indicator function with

$$\delta_{\boldsymbol{C}}(\boldsymbol{p}) = \begin{cases} 0, & \text{if } \boldsymbol{p} \text{ lies on the constraint manifold } \boldsymbol{C} \\ \infty, & \text{otherwise.} \end{cases}$$

Define $\boldsymbol{p_q}$ such that $\psi(\boldsymbol{q}) = d(\boldsymbol{q}, \boldsymbol{p_q}) + \delta_{\boldsymbol{C}}(\boldsymbol{p_q})$. Then obviously $\delta_{\boldsymbol{C}}(\boldsymbol{p_q}) = 0$, meaning that $\boldsymbol{p_q}$ lies on $\boldsymbol{C}$. Together, $\boldsymbol{p_q}$ is the configuration on the constraint manifold $\boldsymbol{C}$ with minimal distance $d(\boldsymbol{q}, \boldsymbol{p_q})$ to current configuration $\boldsymbol{q}$. Consequently, $\psi(\boldsymbol{q})$ measures the distance of $\boldsymbol{q}$ to the constraint manifold $\boldsymbol{C}$.

The authors claim that since the constraint manifolds already capture nonlinearities the need for complicated distance functions d can be relaxed while still achieving visually plausible simulations. In PD, distance measures d are restricted to quadratic functions of the form

$$d(\boldsymbol{q}) = \frac{w}{2}\|\boldsymbol{G}\boldsymbol{q} - \boldsymbol{p}\|_F^2,$$

where $\boldsymbol{G} \in \mathbb{R}^{r\times m}$ for some $r \in \mathbb{N}$.

Note that since $\boldsymbol{q} \in \mathbb{R}^{m\times 3}$, the distance measure d has no dependencies between x-, y- and z-coordinates. This detail demonstrates that restricting to PD energy potentials comes at the cost of generality: Many arbitrary nonlinear elastic potentials, particulary those that have dependencies between x-, y- and z-coordinates, cannot be expressed in terms of PD elastic potentials. However, their structure enables an efficient algorithm for implicit integration, as discussed in the next section.

In summary, PD energy potentials have the following form:

$$\psi(\boldsymbol{q}) = \min_{\boldsymbol{p}} \frac{w}{2}\|\boldsymbol{G}\boldsymbol{q} - \boldsymbol{p}\|_F^2 + \delta_{\boldsymbol{C}}(\boldsymbol{p}).$$

Equations of Motion

Should there be a rough general introduction to implicit integration? Or is it enough to show the resulting equations in the context of the ODE with the equations of motion?

For now, this only has the information required for PD. Check whether more information is required for (x)PBD, ADMM, QN.

Maybe mention other integration schemes than the implicit Euler integration.

The motion of a spatially discretized system with m particles evolving in time according to Newton's laws of motion can be modeled via the following ordinary differential equation (ODE), which will be referred to as Newton's ODE [4, 7, 3]:

$$\begin{aligned}\boldsymbol{q}(t)\prime &= \boldsymbol{v}(t)\\ \boldsymbol{v}(t)\prime &= \boldsymbol{M}^{-1}\boldsymbol{f}(\boldsymbol{q}(t), \boldsymbol{v}(t))\end{aligned}$$

where $\boldsymbol{q}(t)$, $\boldsymbol{v}(t)$, $\boldsymbol{f}(\boldsymbol{q}(t), \boldsymbol{v}(t))$ are the particle positions, particle velocities and forces acting on each particle at time t, respectively, and $\boldsymbol{M}$ is a diagonal matrix with the particle masses as diagonal entries. Depending on the context, either $\boldsymbol{q}(t), \boldsymbol{v}(t), \boldsymbol{f}(\boldsymbol{q}(t), \boldsymbol{v}(t)) \in \mathbb{R}^{m\times 3}$ and $\boldsymbol{M} \in \mathbb{R}^{m\times m}$ or $\boldsymbol{q}(t), \boldsymbol{v}(t), \boldsymbol{f}(\boldsymbol{q}(t), \boldsymbol{v}(t)) \in \mathbb{R}^{3m}$ and $\boldsymbol{M} \in \mathbb{R}^{3m\times 3m}$. $\boldsymbol{q}(t)\prime$ and $\boldsymbol{v}(t)\prime$ are short for $\boldsymbol{D_t q}(t)$ and $\boldsymbol{D_t v}(t)$, respectively. From now on, we write $\boldsymbol{q}$ instead of $\boldsymbol{q}(t)$ for time-dependent quantities for the sake of brevity.

Integration of Newton's ODE

For general nonlinear forces, analytical solutions to Newton's ODE are usually not available. Thus, the ODE needs to be solved numerically. One popular integration scheme for tackling Newton's ODE is implicit Euler integration [4, 7, 3]. Here, the positions and velocities are computed at discrete timesteps via the following update formulas

$$\begin{aligned}\boldsymbol{q}_{n+1} &= \boldsymbol{q}_n + h\boldsymbol{v}_{n+1}\\ \boldsymbol{v}_{n+1} &= \boldsymbol{v}_n + h\boldsymbol{M}^{-1}\boldsymbol{f}(\boldsymbol{q}_{n+1}, \boldsymbol{p}_{n+1})\end{aligned}$$

for some timestep h. Note how $\boldsymbol{q}_{n+1}$ and $\boldsymbol{v}_{n+1}$ appear on both sides of the equations. Consequently, performing implicit Euler integration includes solving a set of nonlinear algebraic equations. It can be shown that implicit Euler integration is unconditionally stable and first-order accurate [5]. However, implicit Euler integration is also known to exhibit numerical damping [13, 3, 8].

By rewriting the first equation as

$$\boldsymbol{v}_{n+1} = \frac{1}{h}(\boldsymbol{q}_{n+1} - \boldsymbol{q}_n)$$

and substituting into the second equation the following equation can be derived

$$\boldsymbol{M}(\boldsymbol{q}_{n+1} - \boldsymbol{q}_n - h\boldsymbol{v}_n) = h^2(\boldsymbol{f}(\boldsymbol{q}_{n+1}, \boldsymbol{v}_{n+1})).$$

We separate forces $\boldsymbol{f}(\boldsymbol{q}, \boldsymbol{v})$ into internal forces $\boldsymbol{f}_{\text{int}}(\boldsymbol{q}, \boldsymbol{p}) = \sum_i \boldsymbol{f}^i_{\text{int}}(\boldsymbol{q}, \boldsymbol{p})$ and external forces $\boldsymbol{f}_{\text{ext}}(\boldsymbol{q}, \boldsymbol{p}) = \sum_i \boldsymbol{f}^i_{\text{ext}}(\boldsymbol{q}, \boldsymbol{p})$. We consider all external forces to be constant. Internal forces are conservative and defined in terms of scalar potential energy functions ψ_i via $\boldsymbol{f}^i_{\text{int}}(\boldsymbol{q}) = -\nabla\psi_i(\boldsymbol{q})$. Together, we have $\boldsymbol{f}(\boldsymbol{q}, \boldsymbol{v}) = \boldsymbol{f}(\boldsymbol{q}) = \boldsymbol{f}_{\text{int}}(\boldsymbol{q}) + \boldsymbol{f}_{\text{ext}} = \sum_i -\nabla\psi_i(\boldsymbol{q}) + \boldsymbol{f}_{\text{ext}}$. Plugging into the equation above, it is

$$\boldsymbol{M}(\boldsymbol{q}_{n+1} - \boldsymbol{q}_n - h\boldsymbol{v}_n) = h^2(\boldsymbol{f}_{\text{ext}} - \sum_i \nabla\psi_i(\boldsymbol{q})).$$

By computing first-order optimality conditions, it is easily verified that the above system of equations is equivalent to the optimization problem

$$\min_{\boldsymbol{q}_{n+1}} \frac{1}{2h^2}\|\boldsymbol{q}_{n+1} - \boldsymbol{s}_n\|^2_F + \sum_i \psi_i(\boldsymbol{q}_{n+1}).$$

where $\boldsymbol{s}_n = \boldsymbol{q}_n + h\boldsymbol{v}_n + h^2\boldsymbol{M}^{-1}\boldsymbol{f}_{\text{ext}}$. This minimization problem whose solution corresponds to the next iteration of the state of the implicit Euler integration is called the variational form of implicit Euler integration [4]. The first and second term of the objective function are called the momentum potential and the elastic potential, respectively. Thus, the minimization problem requires that the solution minimizes the elastic deformation as best as possible while ensuring that the solution is close to following its momentum plus external forces. The weighting between the momentum potential and the elastic potential depends on the particle masses $\boldsymbol{M}$, the timestep h and the material stiffness of the elastic potentials ψ_i. According to Noether's theorem, the solution preserves linear and angular momentum as long as the elastic potentials are rigid motion invariant.

Add some words about how it is often favorable to use the variational formulation because solving an optimization problem is often easier than just solving a system of equations. That is because one can be guided by the objective function.

Projective Implicit Euler Solver

We start by substituting the PD energy potentials into the variatonal form of implicit Euler integration, which yields the following joint optimization problem over the positions $\boldsymbol{q}_{n+1}$ and auxiliary variables $\boldsymbol{p}_i$

$$\min_{\boldsymbol{q}, \boldsymbol{p_i}} \frac{1}{2h^2}\left\|\boldsymbol{M}^{1/2}(\boldsymbol{q} - \boldsymbol{s}_n)\right\|^2_F + \sum_i \|\boldsymbol{G}_i\boldsymbol{q} - \boldsymbol{p_i}\|^2_F + \delta_{\boldsymbol{C}_i}(\boldsymbol{p_i}).$$

We write $\boldsymbol{q} \coloneqq \boldsymbol{q}_{n+1}$ for the sake of brevity.

This optimization problem is optimized using a local/global alternating minimization technique. Local and global steps are carried out sequentially for a fixed number of iterations during each timestep. The local step consists of minimizing the objective function over the auxiliary variables $\boldsymbol{p}_i$ while keeping the positions $\boldsymbol{q}$ fixed. This corresponds to finding the projection points of the current positions onto the constraint manifolds used to define the PD energy potentials. Each constraint has its own, independent set of auxiliary variables, making the optimization amendable to massive parallelization. For each energy potential, we solve the following minimization problem

$$\min_{\boldsymbol{p}_i} \frac{w_i}{2}\|\boldsymbol{G}_i\boldsymbol{q} - \boldsymbol{p}_i\|^2_F + \delta_{\boldsymbol{C}_i}(\boldsymbol{p}_i).$$

In the global step, the minimization problem is optimized over the positions $\boldsymbol{q}$ while keeping the auxiliary variables $\boldsymbol{p}_i$ fixed. This corresponds to moving the positions $\boldsymbol{q}$ according to their momentum and external forces while trying to maintain short distances to the projections points as defined by the distance measures of the PD energy potentials. The optimization problem for the global solve is given by

$$\min_{\boldsymbol{q}} \frac{1}{2h^2}\left\|\boldsymbol{M}^{1/2}(\boldsymbol{q} - \boldsymbol{s}_n)\right\|^2_F + \sum_i \|\boldsymbol{G}_i\boldsymbol{q} - \boldsymbol{p_i}\|^2_F .$$

By design of the PD energy potentials, the objective function of the global optimization problem is quadratic in the positions $\boldsymbol{q}$. Consequently, the minimization can be carried out in a single step by picking $\boldsymbol{q}$ such that the first-order optimality conditions are satisfied [11]. This requires solving the linear system

$$(\frac{\boldsymbol{M}}{h^2} + \sum_i w_i \boldsymbol{G}_i^T \boldsymbol{G}_i)\boldsymbol{q} = \frac{\boldsymbol{M}}{h^2}\boldsymbol{s}_n + \sum_i w_i \boldsymbol{G}_i^T \boldsymbol{p}_i.$$

Since the system matrix is constant as long as the constraints remain unchanged it can be prefactorized at initialization, making efficient global solves possible. Additionally, note that the system can be solved for each coordinate independently, lending itself to parallelization. This results from the fact that $\boldsymbol{q} \in \mathbb{R}^{m\times 3}$, which is also enabled by the structure of the PD energy potentials. The right side needs to be recomputed in every iteration as the projections $\boldsymbol{p}_i$ change during the local optimization steps. The objective function is bounded below and both local and global steps are guaranteed to weakly decrease it. As a result, the optimization converges without additional safeguards, even if non-convex constraint manifolds are used in the energy potentials. An overview over the algorithm is given below.

Algorithm 1 Projective Implicit Euler Solver

$\boldsymbol{s}_n \leftarrow \boldsymbol{q}_n + h\boldsymbol{v}_n + h^2\boldsymbol{M}^{-1}\boldsymbol{f}_{\text{ext}}$
$\boldsymbol{q}_{n+1} = \boldsymbol{s}_n$
for all iterations **do**
  for constraints i **do**
    $\boldsymbol{p}_i = \text{ProjectOnConstraintSet}(\boldsymbol{C}_i, \boldsymbol{q}_{n+1})$
  end for
  $\boldsymbol{q}_{n+1} = \text{SolveLinearSystem}(\boldsymbol{s}_n, \boldsymbol{p}_1, \boldsymbol{p}_2, \ldots)$
end for
$\boldsymbol{v}_{n+1} = (\boldsymbol{q}_{n+1} - \boldsymbol{q}_n)/h$

- Structure of these constraint-based energy potentials allows efficient local/global optimization (block coordinate descent).
- Robust and efficient, often significantly outperforming classical Newton methods.
- Projective dynamics is an extension of [6] from mass-sprint systems to exmploying projection onto constraint sets to simulate general nodal dynamical systems.
- Constraint projection is performed globally here, while it is performed locally in (x)PBD
  - Understand exactly where and how the projection is achieved in (x)PBD
- The global solve (10) has a Hessian matrix according to [4]. Look at the minimization problem with fixed projetion variables again
- [4] show some interesting experiments in their results. Additionally, the listed weaknesses of projective dynamics are pretty interesting.

Unconstrained Optimization

The goal of unconstrained optimization is to find the global minimizer of smooth, but generally nonlinear functions of the form $f\colon \mathbb{R}^n \to \mathbb{R}, n \in \mathbb{N}$, or formally

$$\min_{\boldsymbol{x}} f(\boldsymbol{x}).$$

Here, f is called the objective function. Most algorithms are incapable of finding global minimizers of general nonlinear functions. Instead, these algorithms begin their search at a starting point $\boldsymbol{x}_0$ and then iteratively improve this initial guess until a local minimizer is found [11]. A local minimizer is a point $\boldsymbol{x}^*$ such that there is a neighborhood $\mathcal{N}$ of $\boldsymbol{x}^*$ such that $f(\boldsymbol{x}^*) \leq f(\boldsymbol{x})$ for all $\boldsymbol{x} \in \mathcal{N}$. If the initial guess $\boldsymbol{x}^*$ is close enough to the global minimizer the local minimizer that the algorithm converges to can often coincide with a global minimizer.

Line Search Methods

It can be shown that $\nabla f(\boldsymbol{x}^*) = 0$ if $\boldsymbol{x}^*$ is a local minimizer and f is continuously differentiable in an open neighborhood of $\boldsymbol{x}^*$ [11]. The proof is by contradiction and establishes that if $\nabla f(\boldsymbol{x}^*) \neq 0$, then it is possible to pick a descent direction along which it is possible to decrease the value of the objective function if the step size is picked sufficiently

small. This observation gives rise to the idea of a family of optimization algorithms called line search algorithms [11]: Given the current iterate $\boldsymbol{x}_k$, pick a descent direction $\boldsymbol{p}_k$ and search along this direction for a new iterate $\boldsymbol{x}_{k+1}$ with $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k$. This process is repeated until $\nabla f(\boldsymbol{x}_k)$ is sufficiently close to zero. It is important to note that $\nabla f(\boldsymbol{x})$ does not imply that $\boldsymbol{x}$ is a local minimizer. Instead, $\boldsymbol{x}$ is only guaranteed to be a local minimizer if the second-order optimality conditions are satisfied, which additionally require $\nabla^2 f(\boldsymbol{x})$ to be positive semidefinite [11].

Ideally, α_k is picked such that it is the minimizer of the one-dimensional optimization problem

$$\min_{\alpha_k > 0} f(\boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k).$$

In most cases, it is infeasible to compute α_k exactly. Instead, the idea is to compute an approximation of α_k such that the objective function decreases sufficiently and that α_k is close enough to the true minimizer. Formally, these requirements are captured in the strong Wolfe conditions for step lenghts α_k [11]:

$$\begin{aligned} f(\boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k) &\leq f(\boldsymbol{x}_k) + c_1 \alpha_k \nabla f(\boldsymbol{x}_k) \boldsymbol{p}_k \\ \left| \nabla f(\boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k)^T \boldsymbol{p}_k \right| &\leq c_2 \left| \nabla f(\boldsymbol{x}_k)^T \boldsymbol{p}_k \right| \end{aligned}$$

for some constants $c_1 \in (0, 1), c_2 \in (c_1, 1)$. The first condition is called the sufficient decrease or the Armijo condition and states that the reduction in f should be proportional to both the step length α_k and the directional derivative $\nabla f(\boldsymbol{x}_k \boldsymbol{p}_k)$. Informally, the second condition, known as the curvature condition, ensures that there is no more fast progress to be made along the search direction $\boldsymbol{p}_k$, indicated by the fact that $\left| \nabla f(\boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k)^T \boldsymbol{p}_k \right|$ is already rather small.

Step sizes satisfying the strong Wolfe conditions have the following properties under mild assumptions [11]. Firstly, if $\boldsymbol{p}_k$ is a descent direction, it is possible to find a step size that satisfies the strong Wolfe conditions. Secondly, it can be shown that line search methods where α_k satisfies the strong Wolfe conditions for all k converge to a stationary point $\boldsymbol{x}^*$ with $\nabla f(\boldsymbol{x}^*) = 0$ if the search direction $\boldsymbol{p}_k$ is sufficiently far from orthogonal to the steepest descent direction $\nabla f(\boldsymbol{x}_k)$ for all k. Such line search algorithms are called globally convergent.

Steepest Descent

The most obvious choice for the search direction $\boldsymbol{p}_k$ at iteration k is the steepest descent direction given by

$$\boldsymbol{p}_k = -\nabla f(\boldsymbol{x}_k).$$

This method is called steepest descent. While simple, steepest descent exhibits poor performance, even for simple problems [11]. Its convergence rate is only linear and depends on the eigenvalue distribution of the Hessian $\nabla^2 f(\boldsymbol{x}^*)$ at the local minimizer $\boldsymbol{x}^*$. If the eigenvalues distribution is wide, steepest descent often requires an unacceptably large number of iterations to find a stationary point.

Newton Method

It can be shown that any search direction $\boldsymbol{p}_k$ that makes an angle of strictly less than $\pi/2$ radians with the steepest descent direction $\nabla f(\boldsymbol{x}_k)$ is a descent direction as well [11]. As long as $\boldsymbol{p}_k$ does not get arbitrarily close to orthogonal to $\nabla f(\boldsymbol{x}_k)$, any such $\boldsymbol{p}_k$ can be used in the line search framework. The so called Newton direction $\boldsymbol{p}_k^N$ is a popular choice. It is derived from the second-order Taylor series approximation to $f(\boldsymbol{x}_k + \boldsymbol{p})$ which is given by

$$f(\boldsymbol{x}_k + \boldsymbol{p}) \approx f(\boldsymbol{x}_k) + \boldsymbol{p}^T \nabla f(\boldsymbol{x}_k) + \frac{1}{2} \boldsymbol{p}^T \nabla^2 f(\boldsymbol{x}_k) \boldsymbol{p} =: m_k(\boldsymbol{p}).$$

The model function m_k has a unique minimizer if $\nabla^2 f(\boldsymbol{x}_k)$ is positive definite. In this case, the Newton direction is defined as the unique minimizer $\boldsymbol{p}_k^N$ of m_k, which can be found by setting the derivative of $m_k(\boldsymbol{p})$ to zero:

$$\boldsymbol{p}_k^N = -(\nabla^2 f(\boldsymbol{x}_k))^{-1} \nabla f(\boldsymbol{x_k}).$$

The better the quadratic model function $m_k(\boldsymbol{p})$ approximates $f(\boldsymbol{x}_k + \boldsymbol{p})$ around $\boldsymbol{x}_k$, the more reliable is the Newton direction.

It is easy to show that $\boldsymbol{p}_k^N$ is a descent direction, given that $\nabla^2 f(\boldsymbol{x}_k)$ is positive definite [11]. Otherwise, the Newton direction is not guaranteed to exist, or to be a descent direction if it does. In such cases, the Newton direction cannot be used without modification. Thus, in its naive form, the Newton method is not globally convergent. However, if $\nabla^2 f(\boldsymbol{x}^*)$ is positive definite at a local solution $\boldsymbol{x}^*$ and f is twice differentiable, then $\nabla^2 f(\boldsymbol{x})$ is also positive definite for $\boldsymbol{x} \in \mathcal{N}$ for some neighborhood $\mathcal{N}$ of $\boldsymbol{x}^*$. If we have $\boldsymbol{x}_0 \in \mathcal{N}$ for the starting point of $\boldsymbol{x}_0$ of Newton's method and $\boldsymbol{x}_0$ is sufficiently close to the solution $\boldsymbol{x}^*$ it can be shown that Newton's method with step length $\alpha_k = 1$ converges to $\boldsymbol{x}^*$ with a quadratic rate of convergence under mild conditions [11]. Thus, Newton's method has satisfactory convergence properties close to the solution $\boldsymbol{x}^*$ and the Newton direction $\boldsymbol{p}_k^N$ has a natural step size $\alpha_k = 1$ associated with it. Since $\alpha_k = 1$ often does not satisfy the Wolfe conditions when the current iterate $\boldsymbol{x}_k$ is still far away from the solution $\boldsymbol{x}^*$, line searches are still necessary in Newton's method. However, it is recommended to use $\alpha_k = 1$ as the initial guess as $\alpha_k = 1$ guarantees quadratic convergence once $\boldsymbol{x}_k$ gets sufficiently close to $\boldsymbol{x}^*$.

Despite its favorable convergence properties, Newton's method comes with a couple of disadvantages. Firstly, computing the Hessian matrix $\nabla^2 f(\boldsymbol{x}_k)$ is expensive and error prone. Additionally, a new system

$$\nabla^2 f(\boldsymbol{x}_k)\boldsymbol{p}_k^N = -\nabla f(\boldsymbol{x_k})$$

needs to be solved at every iteration as the Hessian matrix changes with the current iterate $\boldsymbol{x}_k$. For these reasons, while a single Newton iteration often makes quite a lot of progress towards the solution, it takes a significant amount of time to compute. If $\boldsymbol{x}_k \in \mathbb{R}^n$ for some large $n \in \mathbb{N}$, computing the exact Newton iteration can become infeasible, especially for real-time applications. Concomitantly, the memory required to store the Hessian matrix of size $\mathcal{O}(n^2)$ becomes prohibitive.

Quasi-Newton Methods

Due to the shortcomings of Newton's method mentioned above, it can be favorable to simply approximate the Newton direction in order to find an effective search direction while keeping the cost of a single iteration low. Effective Newton approximations can be computed without the need to compute the Hessian $\nabla^2 f(\boldsymbol{x}_k)$ during each iteration [11]. Usually, multiple Quasi-Newton iterations fit into the same time budget as a single Newton iteration. As a result, Quasi-Newton methods can converge to a solution in a shorter amount of time than the Newton method, even though their search directions are not as effective as the exact Newton direction.

In Quasi-Newton methods, search directions of the following form are used

$$\boldsymbol{p}_k = -\boldsymbol{B}_k^{-1}\nabla f(\boldsymbol{x}_k),$$

where $\boldsymbol{B}_k \in \mathbb{R}^{n\times n}$ is positive definite [11]. Note that the Newton direction is a special case of the above with $\boldsymbol{B}_k = \nabla^2 f(\boldsymbol{x}_k)$. Just like for the Newton direction $\boldsymbol{p}_k^N$, a model function m_k that attains its minimum at $\boldsymbol{p}_k = -\boldsymbol{B}_k^{-1}\nabla f(\boldsymbol{x}_k)$ can be defined via

$$m_k(\boldsymbol{p}) = f(\boldsymbol{x}_k) + \nabla f(\boldsymbol{x}_k)^T\boldsymbol{p} + \frac{1}{2}\boldsymbol{p}^T\boldsymbol{B}_k\boldsymbol{p}.$$

As $\boldsymbol{B}_k \neq \nabla^2 f(\boldsymbol{x}_k)$, m_k does not correspond to a second-order Taylor approximation of f around $\boldsymbol{x}_k$ anymore. Instead, $\boldsymbol{B}_k$ is picked such that the gradient of m_k matches the gradient of f at the last two iterates $\boldsymbol{x}_k$ and $\boldsymbol{x}_{k-1}$. Since $\nabla m_k(\boldsymbol{0}) = \nabla f(\boldsymbol{x}_k + \boldsymbol{0}) = \nabla f(\boldsymbol{x}_k)$, the first condition is true independent of $\boldsymbol{B}_k$. The second condition yields

$$\nabla m_k(-\alpha_{k-1}\boldsymbol{p}_{k-1}) = \nabla f(\boldsymbol{x}_k) - \alpha_{k-1}\boldsymbol{B}_k\boldsymbol{p}_{k-1} = \nabla f(\boldsymbol{x}_k - \alpha_{k-1}\boldsymbol{p}_{k-1}) = \nabla f(\boldsymbol{x}_{k-1}).$$

Rearranging gives

$$\boldsymbol{B}_k\boldsymbol{s}_{k-1} = \boldsymbol{y}_{k-1},$$

where $\boldsymbol{s}_{k-1} = \boldsymbol{x}_k - \boldsymbol{x}_{k-1} = \alpha_{k-1}\boldsymbol{p}_{k-1}$. This is called the secant equation. Multiplying both sides from the left with $\boldsymbol{s}_{k-1}^T$ yields the curvature condition given by

$$\boldsymbol{s}_{k-1}^T \boldsymbol{y}_{k-1} > 0,$$

since $\boldsymbol{B}_k$ is positive definite. Note that the curvature condition is not satisfied for arbitrary $\boldsymbol{x}_k, \boldsymbol{x}_{k-1}$ if f is not convex. However, it can be shown that the curvature condition always holds when the step size α_{k-1} satisfies the strong Wolfe conditions [11]. Thus, a proper line search strategy is vital for the viability of Quasi-Newton methods.

Since $\boldsymbol{B}_k$ is positive definite, the secant equation can be written in terms of the inverse $\boldsymbol{H}_k \coloneqq \boldsymbol{B}_k^{-1}$ as

$$\boldsymbol{H}_k \boldsymbol{y}_{k-1} = \boldsymbol{s}_{k-1}$$

and the formula for the new search direction becomes $-\boldsymbol{H}_k \nabla f(\boldsymbol{x}_k)$. The secant equation is not enough to uniquely determine the entries of $\boldsymbol{H}_k$, even if $\boldsymbol{H}_k$ is required to be symmetric positive definite. Thus, the additional requirement that $\boldsymbol{H}_k$ is closest to $\boldsymbol{H}_{k-1}$ according to some norm is imposed. In summary, $\boldsymbol{H}_k$ is picked such that it solves the following constrained minimization problem

$$\min_H \|\boldsymbol{H} - \boldsymbol{H}_{k-1}\|, \text{ subject to } \boldsymbol{H} = \boldsymbol{H}^T \text{ and } \boldsymbol{H}\boldsymbol{y}_{k-1} = \boldsymbol{s}_{k-1}.$$

Using a scale-invariant version of the weighted Frobenius norm gives rise to the popular Broyden- Fletcher-Goldfarb-Shanno (BFGS) algorithm. It is defined via the following update formula for $\boldsymbol{H}_k$

$$\boldsymbol{H}_k = (I - \rho_{k-1}\boldsymbol{s}_{k-1}\boldsymbol{y}_{k-1}^T)\boldsymbol{H}_{k-1}(I - \rho_{k-1}\boldsymbol{s}_{k-1}\boldsymbol{y}_{k-1}^T) + \rho_{k-1}\boldsymbol{s}_{k-1}\boldsymbol{s}_{k-1}^T,$$

where $\rho_{k-1} = 1/(\boldsymbol{s}_{k-1}^T \boldsymbol{y}_{k-1})$. Alternatively, the update formula can be given in terms of $\boldsymbol{B}_k$ as

$$\boldsymbol{B}_k = \boldsymbol{B}_{k-1} - \frac{\boldsymbol{B}_{k-1}\boldsymbol{s}_{k-1}\boldsymbol{s}_{k-1}^T\boldsymbol{B}_{k-q}}{\boldsymbol{s}_{k-1}^T\boldsymbol{B}_{k-1}\boldsymbol{s}_{k-1}} + \frac{\boldsymbol{y}_{k-1}\boldsymbol{y}_{k-1}^T}{\boldsymbol{y}_{k-1}^T\boldsymbol{s}_{k-1}}.$$

Generally, using the formulation in terms of the inverse matrices $\boldsymbol{H}_k$ is preferrable since the computation of the new descent direction can be achieved by simple matrix-vector multiplication instead of solving a linear system if $\boldsymbol{B}_k$ is maintained instead.

While global convergence of the BFGS method cannot be established for general nonlinear smooth functions, it is possible to show that it converges superlinearly if the initial guess $\boldsymbol{x}_0$ is close to the solution $\boldsymbol{x}^*$ and $\alpha_k = 1$ for sufficiently large k [11] under mild conditions. Typically, the BFGS method dramatically outperforms steepest descent and performs comparably to Newton's method on many practical problems.

The behavior of the BFGS method depends on the choice of the initial inverse matrix $\boldsymbol{H}_0$. One obvious choice is $\boldsymbol{H}_0 = \nabla^2 f(\boldsymbol{x}_0)$. However, there is no guarantee that $\nabla^2 f(\boldsymbol{x}_0)$ is positive definite. Additionally, computing even a single inverse matrix can be prohibitively expensive for large problems. Thus, scaled versions of the identity matrix $\gamma I, \gamma \in \mathbb{R}^+$ are often used instead. There is no good general strategy for choosing γ, even though heuristic appraoches are popular. Maybe explain one heuristic, if necessary down the line.

Even though BFGS iterations are typically faster to compute than Newton iterations, the BFGS method is still not suitable for large problems in its naive form. Just like in Newton's method, either $\boldsymbol{H}_k$ or $\boldsymbol{B}_k$ needs to be stored explicitly, which can be infeasible for large-scale problems. While the BFGS update formula using the inverse matrices $\boldsymbol{H}_k$ replaces the need for a matrix factorization with a simple matrix-vector multiplication, $\boldsymbol{H}_k$ and $\boldsymbol{B}_k$ are generally dense, even if $\nabla^2 f(\boldsymbol{x}_k)$ is sparse. This removes the possibility of alleviating storage requirements and speeding up computations via sparse matrix techniques when using the naive BFGS method.

- L-BFGS
- Double recursion

References

[1] David Baraff. Linear-time dynamics using lagrange multipliers. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '96, page 137–146, New York, NY, USA, 1996. Association for Computing Machinery.

[2] David Baraff and Andrew Witkin. Large steps in cloth simulation. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '98, page 43–54, New York, NY, USA, 1998. Association for Computing Machinery.

[3] Jan Bender, Matthias Müller, and Miles Macklin. A Survey on Position Based Dynamics. In Adrien Bousseau and Diego Gutierrez, editors, *EG 2017 - Tutorials*. The Eurographics Association, 2017.

[4] Sofien Bouaziz, Sebastian Martin, Tiantian Liu, Ladislav Kavan, and Mark Pauly. Projective dynamics: Fusing constraint projections for fast simulation. *ACM Trans. Graph.*, 33(4), jul 2014.

[5] Steven C. Chapra and Raymond Canale. *Numerical Methods for Engineers*. McGraw-Hill, Inc., USA, 5 edition, 2005.

[6] Tiantian Liu, Adam W. Bargteil, James F. O'Brien, and Ladislav Kavan. Fast simulation of mass-spring systems. *ACM Trans. Graph.*, 32(6), nov 2013.

[7] Miles Macklin, Matthias Müller, and Nuttapong Chentanez. Xpbd: position-based simulation of compliant constrained dynamics. In *Proceedings of the 9th International Conference on Motion in Games*, MIG '16, page 49–54, New York, NY, USA, 2016. Association for Computing Machinery.

[8] Miles Macklin, Kier Storey, Michelle Lu, Pierre Terdiman, Nuttapong Chentanez, Stefan Jeschke, and Matthias Müller. Small steps in physics simulation. In *Proceedings of the 18th Annual ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '19, New York, NY, USA, 2019. Association for Computing Machinery.

[9] Matthias Müller, Miles Macklin, Nuttapong Chentanez, Stefan Jeschke, and Tae-Yong Kim. Detailed rigid body simulation with extended position based dynamics. *Computer Graphics Forum*, 39(8):101–112, 2020.

[10] Andrew Nealen, Matthias Müller, Richard Keiser, Eddy Boxerman, and Mark Carlson. Physically based deformable models in computer graphics. *Computer Graphics Forum*, 25(4):809–836, 2006.

[11] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, New York, NY, USA, 2e edition, 2006.

[12] Hanan Rubin and Peter Ungar. Motion under a strong constraining force. *Communications on Pure and Applied Mathematics*, 10(1):65–87, 1957.

[13] Martin Servin, Claude Lacoursière, and Niklas Melin. Interactive simulation of elastic deformable materials. *Proc. SIGRAD*, 01 2006.

[14] Demetri Terzopoulos, John Platt, Alan Barr, and Kurt Fleischer. Elastically deformable models. *SIGGRAPH Comput. Graph.*, 21(4):205–214, aug 1987.

[15] Maxime Tournier, Matthieu Nesme, Benjamin Gilles, and François Faure. Stable constrained dynamics. *ACM Trans. Graph.*, 34(4), jul 2015.

[16] Linda R. Petzold Uri M. Ascher, Hongsheng Chin and Sebastian Reich. Stabilization of constrained mechanical systems with daes and invariant manifolds. *Mechanics of Structures and Machines*, 23(2):135–157, 1995.