Project Proposal: Numerical Optimization

Julian Karrer

June 24, 2025

Introduction

Since the topic came up during the Q&A session today, I wanted to quickly elaborate on the problem I am interested in, to get some feedback on what a viable project for the Numerical Optimization course might look like and how to reduce the larger problem to something more feasible. I wanted to quickly give some context and write down some ideas I had for turning it into a smaller project.

SPH basics

Smoothed Particle Hydrodynamics is an interpolation scheme that allows arbitrary scalar or vector-valued field quantities that are known at discrete positions $\vec{\boldsymbol{x}}_i$ to be interpolated smoothly to any position by convolving the field with a Gaussian-like kernel function $W_{ij} = W\left(||\vec{\boldsymbol{x}}_i - \vec{\boldsymbol{x}}_j||, \hbar\right)$ which is compactly supported with radius $\hbar$. The field A_i at $\vec{\boldsymbol{x}}_i$ can then be approximated by a sum over nearby particles $\mathcal{N}_i$ within $\hbar$, and gradients of A can be found by simply summing the easy-to-compute gradients of the kernel function [1]:

$$A_i \approx \sum_{j \in \mathcal{N}_i} A_j V_j W_{ij}$$
$$\nabla A_i \approx \sum_{j \in \mathcal{N}_i} A_j V_j \nabla W_{ij}$$

where V_j is the volume of particle j.

The problem I actually want to solve

A simple, standard problem is simulating a resting water column under hydrostatic pressure, where the higher the column of water that can be simulated without oscillations or volume loss, and the faster, the better.

When using SPH for Lagrangian fluid simulation of incompressible, low-viscosity liquids such as water (`https://youtu.be/fPKtv8QOaYw`), pressure forces are typically dominant. Therefore, operator splitting is usually applied: viscosity, gravity and other external forces are integrated explicitly, and the stiff subproblem of computing pressure values such that each particle is uncompressed is done implicitly. This yields a linear system called the Pressure Poisson equation or PPE [1], which can be writen in matrix form:

$$\mathbb{A}\vec{\boldsymbol{p}} = \vec{\boldsymbol{s}} \tag{1}$$

where source term $\vec{\boldsymbol{s}}$ expresses: "*what is the predicted density at particle i due to gravity, viscosity etc. pushing particles together, compared to some target rest density ρ_0*" and the linear map $(\mathbb{A}\vec{\boldsymbol{p}})_i$ computes for each particle i "*given pressures $\vec{\boldsymbol{p}}$, how much is the density at particle i predicted to change due to pressure accelerations in the direction of the negative pressure gradient pushing particles apart*". When the density error due to external forces $\vec{\boldsymbol{s}}$ is exactly balanced by the density correction caused by pressure forces $\mathbb{A}\vec{\boldsymbol{p}}$, the $\vec{\boldsymbol{p}}$ solves the PPE and the pressure is valid.

$\mathbb{A}$ is symmetric since pressure forces must be symmetric (I am not entirely convinced this the case if particles don't have uniform volumes), and is positive semidefinite. $\mathbb{A}$ represents a discrete Laplacian of the pressure field with some additional factors (time step Δt^2, -1 depending on the formulation etc.) and is typically assumed to be strictly diagonally dominant (I am pretty sure this is wrong in some unfortunate cases, like when tensile instability causes two particles to be in the same spot/too close together).

$\mathbb{A}$ is $N \times N$ with N typically in the millions, so it is never constructed explicitly, although it is extremely sparse since each particle only interacts with about < 30 neighbours in 3D and ~ 10 neighbours in 2D. The diagonal elements can be (painfully) derived, which is why Jacobi iteration is often used to solve this. Without use of AD tools, which is rarely or never attempted, I don't think anyone ever saw what the off-diagonals look like.

Why projected Jacobi annoys me

Typically [2], a relaxed Jacobi iteration is used to solve the linear system. Unfortunately, SPH underestimates the density at the surface of the fluid, since there are fewer neighbouring particles, which means that the density there can drop below rest density ρ_0. Simply solving the system above then leads to clumping at the surface, or no solution at all if the liquid has a lot of free surface area - instead people in practice project each Jacobi iteration to positive values, simply clamping pressures in each iteration to positive values. So really, there is an inequality constraint that $\vec{\boldsymbol{p}} \geq 0$ should hold which is solved rather ad-hoc.

In other words, we find *non-negative* pressures such that only *compression* $\max\left(0, \frac{\rho_i}{\rho_0} - 1\right)$ (and not any density error ρ_i/ρ_0) is prevented up to some tolerance threshold. Unfortunately, this is trivially true if all pressures are extremely high, since if the liquid explodes it is clearly not compressed. In my opinion, this means that the problem is ill-posed and explains why the projected relaxed Jacobi iteration is unstable and explodes when the pressure solver is initialized to a $\vec{\boldsymbol{p}}_0$ that is too high, while it is relatively stable for a cold-started $\vec{\boldsymbol{p}}_0 = \vec{\mathbf{0}}$, which is annoyingly inefficient since time steps are small and pressures don't change much from previous solutions in each time step.

What I think is correct

Instead, I think that the true pressure field one should compute is the *least* pressure field under some norm that prevents compressions. I'm pretty sure that the least pressure field, if pressure at boundaries is zero, leads to the smallest pressure gradients that still enforce incompressibility, which by Gauss's principle of least constraint [3, can be shown to yield] the physically correct pressures. That would lead to an optimization problem:

$$\begin{aligned} &\min_{\vec{\boldsymbol{p}} \in \mathbb{R}^N} ||\vec{\boldsymbol{p}}||_L \\ \text{s.t.} \quad &\vec{\boldsymbol{p}} \geq 0 \\ &\mathbb{A}\vec{\boldsymbol{p}} - s \geq 0 \end{aligned}$$

where as far as I know $L = 1$ and $L = \infty$ would make this an LP. Then, we would truly find the *least*, *positive* pressure that ensures no compression. I don't think the feasible set here is always non-empty, since shocks can occur that cause compressions which cannot be resolved in one time step to lie below a fixed maximum density error, so maybe an exact penalty method which penalizes $\mathbb{A}\vec{\boldsymbol{p}} - s$ as part of the objective is more appropriate, although a sensible weighting might be hard to derive. I also thought about splitting this lexicographically into two problems using slack variables, where first I find the minimum violation $\mathbb{A}\vec{\boldsymbol{p}} - s$ and then minimize the pressure field. Both of these approaches don't seem performance-viable with current general LP solvers, since $\mathbb{A}\vec{\boldsymbol{p}}$ is an extremely large, sparse linear map computed on the GPU. I looked into Nvidia's GPU solver CuOpt, which was released only a few weeks ago, but it seems like a pain to use in this case (it's largely undocumented C code), and not to be performant enough to solve this system in fractions of a second. If I ignore performance, this might be a viable project though?

Why nonlinear nonsmooth conjugate gradients solvers annoy me less

Recently, the slow convergence of the cold-started projected relaxed Jacobi solver was suggested to be improved [4, by Probst & Teschner from our Computer Graphics group], who built on work by [5, Silcowitz-Hansen et. al] (whose paper reads more like a numerical optimization paper). It is done by interpreting the difference in Jacobi iterates as a residual function $R(\vec{\boldsymbol{p}}^k)$ and bringing it to zero in a least-squares minimization using Fletcher-Reeves type nonlinear nonsmooth conjugate gradients (the algorithm is described in [4, Appendix B, page 27]). This is equivalent to solving the linear complementary problem [4]:

$$0 \leq \vec{\boldsymbol{p}} \quad \perp \quad \vec{\boldsymbol{\rho}}_{error} \geq 0$$

I'd be interested in implementing this, testing if other non-smooth CG methods like those by [6] Hestenes-Stiefel, Polak-Ribière, Dai-Yuan etc. can yield improvements or investigating other Newton-Type methods to accelerate the inner Jacobi solver, so long as it only requires the approximate gradient information derived from the Jacobi iterates. I already have a working Jacobi-based SPH pressure solver, so this is feasible workload-wise.

It would be even nicer to find a scalable algorithm that can efficiently solve the full LP instead of this LCP, since the LCP approach shares the problems of the Jacobi sub-solver and I ideally want a solver that can be aggressively warm-started.

What do I do?

For this project, I could try (sorted by the keywords in the project guidlines):

- (CLASS OF PROBLEM) Implementing the algorithm by Probst & Teschner and comparing it to the regular Jacobi variant of SPH pressure solver for fluids. This might be hard to explain in 10 minutes but very interesting and doable since I already have the Jacobi subsolver.
- (CLASS OF PROBLEM) Checking alternative NNCG methods or other Newton-type methods for solving this LCP.
- (ALGORITHM) Exploring other Krylov-subspace methods like GMRES with fewer requirements on $\mathbb{A}$ to solve the linear system $\mathbb{A}\vec{\boldsymbol{p}} - \vec{\boldsymbol{s}} = 0$ / minimize the corresponding quadratic

Or perhaps something smaller:

- (ALGORITHM) Getting a feel for CG methods in general by implementing CG on the GPU and attempting to integrate inequality constraints for an easier problem as you suggested. I joyfully noticed that the hanging chain example in exercise sheet 5 seems to have a symmetric, positive definite matrix $\mathbb{Q}$ I could try this on. However, I am not convinced that adding inequality constraints using barrier functions will be easy to do or not mess up the convergence of CG? As far as I know even the nonlinear CG variants only help if the minimized function is close to a quadratic form near the solution?
- (ALGORITHM) Implementing and comparing nonlinear CG algorithms for suitable toy problems. This should be doable and fun. I imagined something similar to the illustrations [7, on page 44 of this report]. A downside is that if the implementation is more challenging than I tought I could be stuck.
- (APPLICATION) Implementing a basic 2D SPH fluid solver in Python and using CasADi to solve the LP I envisioned for minimum non-negative pressure values. This might be slow, but I would be interested in seeing how the solution compares to that of the usual solvers and if there is any improvement in how robustly the LP approach handles sudden shocks or how smooth the solution is temporally in a dynamic setting. This should certainly be feasible and interesting.

Which of these do you think would be best? Do any of these suggestions seem unfitting for the course project? Do you think the barrier function in CG will work? Can I swap topics later if I notice later that I chose one that is too difficult or yields no good insights? Can I just choose any of these?

Bibliography

[1] D. Koschier, J. Bender, B. Solenthaler, and M. Teschner, "A survey on sph methods in computer graphics," *Computer Graphics Forum*, vol. 41, no. 2, pp. 737–760, 2022. DOI: https://doi.org/10.1111/cgf.14508. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.14508.

[2] M. Ihmsen, J. Cornelis, B. Solenthaler, C. Horvath, and M. Teschner, "Implicit incompressible sph," *IEEE Transactions on Visualization and Computer Graphics*, vol. 20, no. 3, pp. 426–435, Mar. 2014, ISSN: 1941-0506. DOI: 10.1109/TVCG.2013.105.

[3] H.Taha (حيا يا تام تاها.), C. Gonzalez, and M. Shorbagy (آسوربژي ما هامماد), "A minimization principle for incompressible fluid mechanics," *Physics of Fluids*, vol. 35, no. 12, p. 127 110, Dec. 2023, ISSN: 1070-6631. DOI: 10.1063/5.0175959. [Online]. Available: https://doi.org/10.1063/5.0175959.

[4] T. Probst and M. Teschner, "Unified pressure, surface tension and friction for sph fluids," *ACM Trans. Graph.*, vol. 44, no. 1, Dec. 2024, ISSN: 0730-0301. DOI: 10.1145/3708034. [Online]. Available: https://cg.informatik.uni-freiburg.de/publications/2024_TOG_unifiedPressureSurfaceTensionFriction_v2.pdf.

[5] M. Silcowitz-Hansen, S. Niebe, and K. Erleben, "A nonsmooth nonlinear conjugate gradient method for interactive contact force problems," *Vis. Comput.*, vol. 26, no. 6–8, pp. 893–901, Jun. 2010, ISSN: 0178-2789. DOI: 10.1007/s00371-010-0502-6. [Online]. Available: https://doi.org/10.1007/s00371-010-0502-6.

[6] W. Hager, "A survey of nonlinear conjugate gradient method," vol. 2, Jan. 2006. [Online]. Available: https://www.cmor-faculty.rice.edu/~zhang/caam554/pdf/cgsurvey.pdf.

[7] J. R. Shewchuk, "An introduction to the conjugate gradient method without the agonizing pain," USA, Tech. Rep., 1994. [Online]. Available: https://www.cs.cmu.edu/~quake-papers/painless-conjugate-gradient.pdf.