Université de Montréal

Quasi Second-Order Methods for PDE-Constrained Forward and Inverse Problems

par

Jonas Zehnder

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Thèse présentée en vue de l'obtention du grade de
Philosophiæ Doctor (Ph.D.)
en Informatique

mai 2021

Université de Montréal

Faculté des arts et des sciences

Cette thèse intitulée

Quasi Second-Order Methods for PDE-Constrained Forward and Inverse Problems

présentée par

Jonas Zehnder

a été évaluée par un jury composé des personnes suivantes :

Mikhail Bessmeltsev
(président-rapporteur)

Bernhard Thomaszewski
(directeur de recherche)

Derek Nowrouzezahrai
(membre du jury)

Alec Jacobson
(examinateur externe)

Jacques Bélair
(représentant du doyen de la FESP)

Résumé

La conception assistée par ordinateur (CAO), les effets visuels, la robotique et de nombreux autres domaines tels que la biologie computationnelle, le génie aérospatial, etc. reposent sur la résolution de problèmes mathématiques. Dans la plupart des cas, des méthodes de calcul sont utilisées pour résoudre ces problèmes. Le choix et la construction de la méthode de calcul ont un impact important sur les résultats et l'efficacité du calcul. La structure du problème peut être utilisée pour créer des méthodes, qui sont plus rapides et produisent des résultats qualitativement meilleurs que les méthodes qui n'utilisent pas la structure. Cette thèse présente trois articles avec trois nouvelles méthodes de calcul s'attaquant à des problèmes de simulation et d'optimisation contraints par des équations aux dérivées partielles (EDP).

Dans le premier article, nous abordons le problème de la dissipation d'énergie des solveurs fluides courants dans les effets visuels. Les solveurs de fluides sont omniprésents dans la création d'effets dans les courts et longs métrages d'animation. Nous présentons un schéma d'intégration temporelle pour la dynamique des fluides incompressibles qui préserve mieux l'énergie comparé aux nombreuses méthodes précédentes. La méthode présentée présente une faible surcharge et peut être intégrée à un large éventail de méthodes existantes. L'amélioration de la conservation de l'énergie permet la création d'animations nettement plus dynamiques.

Nous abordons ensuite la conception computationelle dont le but est d'exploiter l'outils computationnel dans le but d'améliorer le processus de conception. Plus précisément, nous examinons l'analyse de sensibilité, qui calcule les sensibilités du résultat de la simulation par rapport aux paramètres de conception afin d'optimiser automatiquement la conception. Dans ce contexte, nous présentons une méthode efficace de calcul de la direction de recherche de Gauss-Newton, en tirant parti des solveurs linéaires directs épars modernes. Notre méthode réduit considérablement le coût de calcul du processus d'optimisation pour une certaine classe de problèmes de conception inverse.

Enfin, nous examinons l'optimisation de la topologie à l'aide de techniques d'apprentissage automatique. Nous posons deux questions : Pouvons-nous faire de l'optimisation topologique sans maillage et pouvons-nous apprendre un espace de solutions d'optimisation topologique.

Nous appliquons des représentations neuronales implicites et obtenons des résultats structurellement sensibles pour l'optimisation topologique sans maillage en guidant le réseau neuronal pendant le processus d'optimisation et en adaptant les méthodes d'optimisation topologique par éléments finis. Notre méthode produit une représentation continue du champ de densité. De plus, nous présentons des espaces de solution appris en utilisant la représentation neuronale implicite.

Mots-clés: simulation de fluides, advection-réflexion, analyse de sensibilité, Gauss-Newton, conception computationnelle, représentations neuronales implicites, optimisation topologique, espace des solutions

Abstract

Computer-aided design (CAD), visual effects, robotics and many other fields such as computational biology, aerospace engineering etc. rely on the solution of mathematical problems. In most cases, computational methods are used to solve these problems. The choice and construction of the computational method has large impact on the results and the computational efficiency. The structure of the problem can be used to create methods, that are faster and produce qualitatively better results than methods that do not use the structure. This thesis presents three articles with three new computational methods tackling partial differential equation (PDE) constrained simulation and optimization problems.

In the first article, we tackle the problem of energy dissipation of common fluid solvers in visual effects. Fluid solvers are ubiquitously used to create effects in animated shorts and feature films. We present a time integration scheme for incompressible fluid dynamics which preserves energy better than many previous methods. The presented method has low overhead and can be integrated into a wide range of existing methods. The improved energy conservation leads to noticeably more dynamic animations.

We then move on to computational design whose goal is to harnesses computational techniques for the design process. Specifically, we look at sensitivity analysis, which computes the sensitivities of the simulation result with respect to the design parameters to automatically optimize the design. In this context, we present an efficient way to compute the Gauss-Newton search direction, leveraging modern sparse direct linear solvers. Our method reduces the computational cost of the optimization process greatly for a certain class of inverse design problems.

Finally, we look at topology optimization using machine learning techniques. We ask two questions: Can we do mesh-free topology optimization and can we learn a space of topology optimization solutions. We apply implicit neural representations and obtain structurally sensible results for mesh-free topology optimization by guiding the neural network during optimization process and adapting methods from finite element based topology optimization. Our method produces a continuous representation of the density field. Additionally, we present learned solution spaces using the implicit neural representation.

Keywords: fluid simulation, advection-reflection, sensitivity analysis, Gauss-Newton, computational design, implicit neural representations, topology optimization, solution space

Contents

List of Tables

List of Figures

List of Symbols and Abbreviations

ADMM	Alternating direction method of multipliers
ALM	Augmented Lagrangian method
APIC	Affine particle-in-cell method
BDF	Backward differentiation formula
BFECC	Back and forth error compensation and correction
BFGS	Broyden–Fletcher–Goldfarb–Shanno algorithm
BiCGSTAB	Biconjugate gradient stabilized method
CAD	Computer-aided design
CG	Conjugate gradient method
CNN	Convolutional neural network
DFP	Davidon–Fletcher–Powell method
DGN	Dense Gauss-Newton
DoF	Degree of Freedom
FEM	Finite element method
FLIP	Fluid-implicit-particle
GGN	Generalized Gauss-Newton
ILUT	Incomplete lower–upper decomposition
KKT-conditions	Karush–Kuhn–Tucker conditions
L-BFGS	Limited memory Broyden–Fletcher–Goldfarb–Shanno algorithm
LDL^T	Cholesky decomposition
MC	MacCormack method
MC + R	MacCormack method with reflection
MMSE	Moving mean squared error
MLP	Multilayer perceptrons
PDE	Partial differential equation
RBF	Radial basis function
RMS error	Root-mean-square error
SF	Stable fluids
SGD	Stochastic gradient descent

SGN	Sparse Gauss-Newton
SIMP	Solid isotropic material with penalization
SIREN	Sinusoidal representation networks
SR1	Symmetric rank-one method
SPH	Smoothed-particle hydrodynamics
SQP	Sequential quadratic programming
TO	Topology optimization
QUICK	Quadratic upstream interpolation for convective kinematics

Acknowledgments

I would like to express my greatest gratitude to my advisor Professor Bernhard Thomaszewski for the opportunities, support, help and discussions during my PhD.

I would also like to thank all my co-authors, especially Stelian Coros, Rahul Narain and Yue Li, without them this research would not have been possible and working with them was a great pleasure.

Additionally, I would like to express my gratitude to Pierre Poulin, Mikhail Bessmeltsev and all the other members of the Ligum lab for the valuable discussions and making the workspace so compelling. I also thank my friends and family for their support, I could not imagine doing this without them.

I would also like to thank the Thesis Jury for spending their valuable time on this thesis process and giving much appreciated feedback.

Finally, I would like to thank for the funding provided by Google and the Natural Sciences and Engineering Research Council (NSERC).

Chapter 1

Introduction

Crafting designs and animations is at the heart of what engineers, visual artists and many other practitioners do. This demand has led to much research and development for better tools and computers. Originally the design and animation work was done using analytic methods or by running computational methods by hand, i.e., first a method was devised that would compute the desired numerical result and then humans would carry the method out using pen and paper. However, such an approach has the severe limitation of the computational speed of humans. In recent decades running computational methods on silicone hardware have superseded manual approaches in many areas. An early example of this is the Sydney Opera House for which structural analysis was done for the shell roof using computers [Jones et al., 2006].

But structural engineering is only one of many applications nowadays where computational models are used to create better results faster. Whether it is visual effects, animatronics, sculptures or other intricate designs, these computational models empower users to bring their ideas to life. Without computational methods, the work involved would be too time consuming. However, even with great advances in hardware power, solving certain problems using computational methods remains overly costly. The computational method's qualities have a large impact on accuracy and robustness which directly relates to its usefulness.

Qualitatively better results can be obtained at lower cost by using the structure of the problem at hand. A classic example of this is symplectic integration which preserves energy and momentum better over time. A generalization of this is geometric integration, which uses the underlying structure of the problem to preserve geometric properties better and thus are of special interest to many areas of science. Similarly for optimization problems, more efficient optimization methods can be derived by using the structure of the involved quantities, phenomena, objectives, etc., resulting in faster convergence.

In this thesis we present multiple computational methods related to two different topics, i.e., constrained simulation for fluids and constrained optimization. Hence, we first give two

separate introductions and motivations to these two topics. Then three articles are presented, each with its own introduction, related work, and result sections as submitted for publication. Finally, we make conclusions based on the three articles.

1.1. Constrained Simulation for Incompressible Fluids

Physics simulations are ubiquitous in visual effects nowadays. They help artists create believable animations without the need of manually key framing every moving element in a scene. These simulations include many different types of phenomena such as fluids, rigid bodies, elastic bodies, light and many more. To keep running costs as low as possible these phenomena are modeled differently and often are simulated with separate numerical methods. For visual effects one of the most important aspects is plausibility; do the movement and appearance match the audience's expectation. Plausible results are created by applying or approximating the laws of physics. One common approach of creating a model is by first modeling the phenomenon using time-dependent partial differential equations (PDEs). Time-dependent PDEs tell us how the state of a system changes over time. They are formulated with a notion of a continuous system in mind, but computers currently cannot run continuum-based computations. Thus, the model has to be discretized. This includes choosing how to represent the state of the system and how to advance the state in time.

Liquids and smoke are often fully or partially modeled using the incompressible Navier-Stokes equations:

$$\begin{aligned} \frac{\partial \mathbf{u}}{\partial t} &= -(\mathbf{u} \cdot \nabla)\mathbf{u} - \nabla p + \nu \nabla^2 \mathbf{u} + \mathbf{f} \\ \nabla \cdot \mathbf{u} &= 0 \end{aligned} \tag{1.1.1}$$

where $\mathbf{u}(\mathbf{x})$ is the velocity vector of the fluid at position $\mathbf{x}$. The first set of equations tells us how the velocity field $\mathbf{u}$ evolves in time, including advection and forces (pressure p, viscous and external forces $\mathbf{f}$). The second set of equations constrains the evolution of the pressure force ∇p to act on the velocity field such that the velocity field stays divergence free. Divergence measures at a specific point in space how much the field $\mathbf{u}$ acts as a source, positive divergence, or a sink, negative divergence. Incompressibility is therefore synonymous with the velocity field having zero divergence inside the fluid.

There are many ways in which the Navier-Stokes equations can be discretized. Here we focus on one common way in computer graphics, which is the advection-projection method. The advection-projection method has two main steps as the name suggests. The advection step moves material according to its velocity, but this step introduces errors in the velocity field in terms of divergence. The projection step restores zero divergence, making sure that the result looks like an incompressible fluid.

However, this method has a lot of numerical diffusion, which can give the fluid an overly viscous look. There are two main reasons for the viscous look in this scheme. First, the

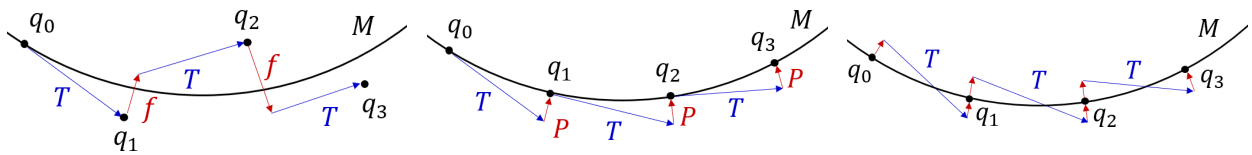


Figure 1.1 – Three possible ways of doing time integration on a constraint manifold: Left: A penalty force f is added to an unconstrained time integrator T which pulls the state q towards the constraint manifold M, e.g., applied in weakly compressible smoothed particle hydrodynamics [Metaxas and Popovic, 2007]. Middle: Advection-projection method [Chorin, 1968; Stam, 1999] where a projection method P is applied after each unconstrained step. Right: Symmetric projection method which adds similar perturbations at the beginning and end of a symmetric integrator T. Symmetric methods are known to preserve the structure of the dynamics well [Hairer et al., 2006].

semi-Lagrangian advection step introduces artificial diffusion due to it relying on interpolation. And second, the projection step discards divergent velocity modes and this directly removes energy [Zhang et al., 2015]. We focus on the second problem in our article in Chapter 2.

As a straightforward solution, one might try to reduce the time step in the integration method, which would reduce the energy loss in the projection step. However as this is only a first order method, it can require a very small time step, which would be computationally very costly and could increase the artificial diffusion from the advection step on the other hand, making this often impractical.

To reduce the energy loss due to the projection, we were interested in applying a second-order symmetric time integration method due to their success in other applications. However, in our tests they were unstable when combined with a semi-Lagrangian advection. Therefore, we propose an alternative solution called advection-reflection, which is quasi second-order in the sense that it is inspired by the structure of a second-order symmetric method and reduces divergence and its associated energy loss to a large degree without the instabilities we have observed with fully symmetric methods.

1.2. Computational Design using Constrained Optimization

Humans have been designing and fabricating objects for a variety of reasons for millennia. Classical manufacturing methods are very efficient and optimized for mass production at low costs and high reliability. However, these methods have a long implementation time from design to manufacturing.

Additive manufacturing brings many new fabrication processes. It allows for faster prototyping, shortening the time to market. Increasing the complexity of the geometry comes with little additional cost. But additive manufacturing does not come without its problems. Often the prototypes from additive manufacturing do not have the same material properties

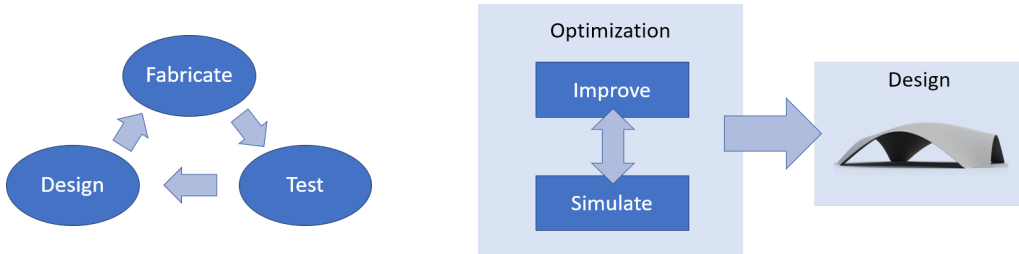


Figure 1.2 – Left: A classical way of creating designs. Each test requires a new physical prototype. Right: An optimization-in-the-loop approach where the simulation acts as PDE constraints on the optimization.

as the ones fabricated with the large-scale manufacturing process. Wrong conclusions can therefore be drawn about the design's performance.

Virtual prototyping is an answer to these issues. It involves validating the design virtually, which is useful and necessary in today's fast moving markets. Using simulation, engineers can test thousands of designs for their performance before even manufacturing a single prototype, thus reducing its development cost. This allows the engineer to improve the design before manufacturing a prototype and thus reducing cost.

In practice it is difficult to estimate the impact of design parameters since the result of the simulation follows the laws of physics and the complexity arising from it. This leads to the requirement of expert knowledge, which takes a long time to acquire. Since modern manufacturing techniques allow for more complex geometries, the number of design parameters may be too large for manual adjustment, even for an expert.

For many applications, the objectives are known. Such objectives can include accuracy, failure resistance and efficiency. Using the objectives and the simulation, a performance-driven design process is possible. Every run of the simulation maps the design to its performance, allowing the engineer to easily select the best design.

For this reason, we want to augment the human designer with optimization-in-the-loop tools. For example, the user could define requirements for the design while the optimization takes care of unrelated design variables and load case parameters. However, to not hamper the design process the optimization needs to be fast enough. Thus, one of our goals is to make the optimization faster. Optimization problems derived for inverse design are very large in practice, requiring the algorithms to be scalable. The methods must be able to handle abstract goals and constraints. To this end we explored and present two new numerical methods in Chapters 3 and 4 to tackle this interesting class of problems.

Chapter 2

An Advection-Reflection Solver for Detail-Preserving Fluid Simulation

by

Jonas Zehnder[1], Rahul Narain[2,3], and Bernhard Thomaszewski[1]

(1) Université de Montréal
(2) University of Minnesota
(3) Indian Institute of Technology Delhi

Publication

This article was submitted to and accepted for publication in ACM Transactions on Graphics (TOG), 37(4), 85. It was presented at the ACM SIGGRAPH 2018 (Vancouver) conference. The article was reformatted to fit the style of this thesis.

Jonas Zehnder was in charge of creating the numerical results for the most part. Jonas Zehnder has presented the method at the conference. Rahul Narain and Bernhard Thomaszewski were in charge of the ideas and writing.

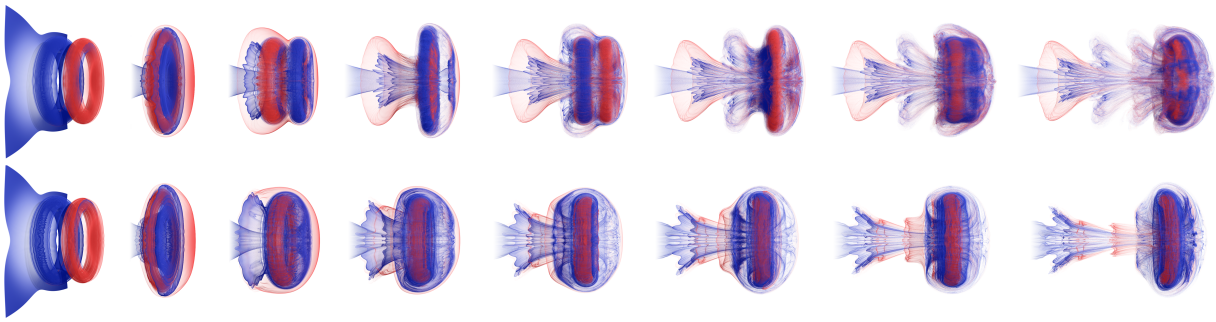


Figure 2.1 – Our new reflection solver applied to a vortex leap-frogging problem (*top row*). During 10s of simulation the two vortex rings move through each other multiple times and stay well separated. By contrast, in a standard advection-projection method with MacCormack advection (*bottom row*), the two vortices merge immediately and never separate afterwards.

Abstract. Advection-projection methods for fluid animation are widely appreciated for their stability and efficiency. However, the projection step dissipates energy from the system, leading to artificial viscosity and suppression of small-scale details. We propose an alternative approach for detail-preserving fluid animation that is surprisingly simple and effective. We replace the energy-dissipating projection operator applied at the end of a simulation step by an *energy-preserving reflection* operator applied at mid-step. We show that doing so leads to two orders of magnitude reduction in energy loss, which in turn yields vastly improved detail-preservation. We evaluate our reflection solver on a set of 2D and 3D numerical experiments and show that it compares favorably to state-of-the-art methods. Finally, our method integrates seamlessly with existing projection-advection solvers and requires very little additional implementation.
Keywords: fluid simulation, advection, reflection, energy conservation

2.1. Introduction

Advection-projection methods are widely used for fluid animations in computer graphics. Splitting mass transport and conservation into different steps allows for stable and efficient integration, and advances in higher-order advection schemes (e.g. [Selle et al., 2008]) have greatly reduced the well-known numerical diffusion caused by the semi-Lagrangian advection step. However, the splitting of the time integration scheme itself induces numerical dissipation, as kinetic energy is transferred to divergent modes during advection and then lost after projection. This numerical dissipation manifests as rapid decay of large vortices and leads to suppression of small-scale swirling motion. Since visual complexity is a central goal in fluid animation, much effort has been spent on combating numerical dissipation: apart from higher-order advection schemes mentioned above, energy-preserving integration [Mullen et al., 2009], a posteriori correction of the velocity field [Fedkiw et al., 2001; Zhang et al., 2015], and injection of procedurally-generated detail [Kim et al., 2008b] are among the strategies that have been pursued so far.

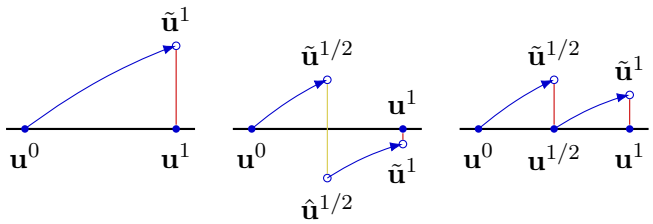


Figure 2.2 – A geometric interpretation of our method. *Left:* In a standard advection-projection solver, projection to the divergence-free subspace causes kinetic energy loss (red). *Middle:* Our reflection solver uses an energy-preserving *reflection* (yellow) halfway through the advection step, dramatically reducing the energy loss caused by the final projection. Our method has effectively identical computational cost to an advection-projection solver with half the time step *(right)*, but loses less energy.

In this work, we propose an alternative approach to detail-preserving fluid animation that is surprisingly simple and effective: we replace the energy-dissipating projection operator applied at the end of a simulation step by an *energy-preserving reflection* operator applied at mid-step; see Fig 2.2. We show that doing so leads to an order of magnitude reduction in divergent kinetic energy, which in turn leads to vastly improved preservation of vortices and small-scale detail. This analysis also exposes our method to be first-order structurally symmetric, which motivates an analogy to (and comparison with) symmetric projection methods for structure-preserving integration of constrained mechanical systems [Hairer et al., 2006]. We show that even the simplest methods from this category are computationally much more expensive and less stable. By contrast, our method preserves the appreciable splitting property of advection-projection methods while offering energy and detail preservation similar to fully-symmetric methods.

Our method integrates seamlessly with existing advection-projection solvers and is agnostic to the choice of advection scheme and pressure discretization. Furthermore, it uses the basic advection and projection steps as primitives, and therefore requires very little additional implementation. We evaluate our reflection solver on an extensive set of 2D and 3D examples and compare its behavior to a number of alternative methods. The results of these comparisons indicate that, for equal computational costs, our method leads to vastly improved energy and vorticity preservation.

2.2. Related Work

Our review of related works focuses primarily on Eulerian fluid simulation methods, as our approach does not apply to Lagrangian particle-based methods like SPH [Ihmsen et al., 2014]. Also, we will only discuss schemes for solving the core Navier-Stokes equations, omitting the diversity of techniques in graphics for artificially injecting detail such as vorticity confinement

[Fedkiw et al., 2001], vortex particles [Selle et al., 2005], and turbulence synthesis [Thuerey et al., 2013].

Most Eulerian methods in graphics follow a Chorin-style advection-projection scheme [Chorin, 1968] introduced to graphics by Stam [1999]. The effectiveness of the advection-projection approach stems from three main ingredients: (i) operator splitting decouples the pressure term from the remaining inertial and internal forces, (ii) semi-Lagrangian advection [Robert, 1981] permits large time steps unimpeded by the CFL condition, and (iii) staggered grids [Harlow and Welch, 1965; Foster and Metaxas, 1996] allow for accurate computation of the pressure projection. While this approach is unconditionally stable, it exhibits noticeable energy loss over time even for inviscid flows. Small-scale vortices and turbulent flows tend to decay especially rapidly, leading to a loss of visually interesting detail. Much work in computer graphics has focused on minimizing this numerical dissipation.

As discussed in the introduction, there are two main reasons for the loss of energy in an advection-projection method: the discretization error in the semi-Lagrangian advection step, which manifests as artificial diffusion, and the splitting error caused by decoupling of the advection and projection steps. To reduce the diffusion in semi-Lagrangian advection, Fedkiw et al. [2001] and Kim et al. [2008a] have proposed higher-order interpolation schemes to improve spatial accuracy. Kim et al. [2005, 2007] introduced the BFECC method which performed multiple backward and forward advection steps to correct both spatial and temporal error. This approach was simplified by Selle et al. [2008], who proposed a semi-Lagrangian variation of the MacCormack method and demonstrated second-order accuracy in space and time. Molemaker et al. [2008] proposed the use of the QUICK advection scheme [Leonard, 1979] for low-dissipation advection, although it is limited by the CFL condition for stability. In this context, hybrid particle-and-grid methods like FLIP [Zhu and Bridson, 2005] and APIC [Jiang et al., 2015] are very attractive as they exhibit little numerical diffusion of this form, because they track the advected quantities on Lagrangian particles which are largely unaffected by the grid interpolation.

In contrast to the improvements in low-diffusion advection schemes, much less attention has been paid to the error introduced by the splitting scheme itself. In graphics, this has been pointed out by Elcott et al. [2007] and Zhang et al. [2015], who noted that semi-Lagrangian advection transfers energy into divergent modes which are then annihilated by the projection step. This is true even for the FLIP and APIC methods, which employ the same advection-projection splitting. Elcott et al. instead adopted the vorticity formulation of the fluid equations, in which the primary variable is the vorticity rather than the velocity field. Other vorticity-based methods in graphics include Park and Kim [2005]; Angelidis and Neyret [2005]; Weißmann and Pinkall [2010]. Since the vorticity representation automatically yields a divergence-free velocity field, such methods do not require a projection step, and consequently do not suffer the associated energy

loss. Particularly notable is the method of Mullen et al. [2009], which is time-reversible and offers exact energy preservation through the use of a symplectic integrator, albeit at the cost of requiring the solution to a nonlinear system at each time step. Nevertheless, advection-projection methods continue to enjoy widespread use in industry and academic research, possibly due to their relative simplicity, efficiency, and flexibility in comparison to vorticity methods [Zhang et al., 2015]. Therefore, we believe that advances in energy preservation for advection-projection methods are still highly desirable.

Remaining within the advection-projection framework, Zhang et al. [2015] counteracted the artificial dissipation caused by the projection step by explicitly tracking the lost vorticity and re-injecting it into the fluid. Thus, their work seeks to preserve vorticity but not necessarily the energy of the fluid. In contrast, we propose a simple modification to the splitting scheme which reduces the projection error without requiring explicit tracking and correction. We also provide a simple proof that our scheme preserves kinetic energy to a higher degree than traditional advection-projection methods.

2.3. Theory

To set the stage for our theoretical developments, we start with a minimal review of advection projection methods before we introduce our reflection solver. For context and comparison, we subsequently introduce two fully-symmetric projection methods.

For notational convenience, we will refer to the velocity field at the beginning and end of the time step as $\mathbf{u}^0$ and $\mathbf{u}^1$ respectively, instead of $\mathbf{u}^n$ and $\mathbf{u}^{n+1}$. We will also adopt the convention that divergence-free velocity fields are undecorated, e.g. $\mathbf{u}^{1/2}$, while velocity fields with nonzero divergence are denoted $\tilde{\mathbf{u}}$ (before projection) or $\hat{\mathbf{u}}$ (after reflection).

2.3.1. Advection-Projection Solvers

The starting point for our developments are the inviscid, incompressible Navier-Stokes equations

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} = -\frac{1}{\rho}\nabla p + \mathbf{f} \tag{2.3.1}$$

$$\nabla \cdot \mathbf{u} = \mathbf{0} \ , \tag{2.3.2}$$

where $\mathbf{u}$ is the continuous velocity field of the fluid, p is the pressure, ρ the density and $\mathbf{f}$ denote external forces such as buoyancy and gravity. Advection-projection methods discretize

the velocities on a Eulerian grid and split the equations into three (or more) different steps:

$$\begin{aligned} \text{Advection} \qquad & \tilde{\mathbf{u}}^1 = A(\mathbf{u}^0; \mathbf{u}^0, \Delta t) \\ \text{Forcing} \qquad & \tilde{\mathbf{u}}^1 \mathrel{+}= \Delta t\, \mathbf{f} \\ \text{Projection} \qquad & \mathbf{u}^1 = P\tilde{\mathbf{u}}^1 \ . \end{aligned}$$

In the above expressions, $A(\mathbf{u}; \mathbf{u}^0, \Delta t)$ is an advection operator that implements a semi-Lagrangian discretization of the advection equation, $\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u}^0 \cdot \nabla)\mathbf{u} = \mathbf{0}$. Furthermore, P is a projection operator that maps a given velocity field to its closest divergence-free field (under the kinetic energy metric). This operator uses the Helmholtz-Hodge decomposition, which splits any vector field $\mathbf{u} = \mathbf{v} + \vec{w}$ into a curl-free part $\mathbf{v}$ and a divergence-free part $\vec{w}$. P simply discards the curl-free part, $P\mathbf{u} = \vec{w}$, by solving a Poisson problem.

For conciseness, we will neglect the forcing step in the following and focus on the central advection and projection steps. In practice, we apply external forces immediately after each advection.

Lemma 2.3.1. *If a vector field $\tilde{\mathbf{u}}$ has divergence $\Theta(\Delta t^k)$, the kinetic energy loss due to projection, $\frac{1}{2}\|\tilde{\mathbf{u}}\|^2 - \frac{1}{2}\|P\tilde{\mathbf{u}}\|^2$, is $\Theta(\Delta t^{2k})$.*

Proof. The pressure projection can be interpreted as decomposing the post-advection velocity $\tilde{\mathbf{u}}$ into two orthogonal components, $\tilde{\mathbf{u}} = P\tilde{\mathbf{u}} + \mathbf{v}$, where $\mathbf{v}$ is the curl-free part. Due to the orthogonality of the Helmholtz-Hodge decomposition, we have $\|\tilde{\mathbf{u}}\|^2 = \|P\tilde{\mathbf{u}}\|^2 + \|\mathbf{v}\|^2$. Therefore, the energy loss $\frac{1}{2}\|\tilde{\mathbf{u}}\|^2 - \frac{1}{2}\|P\tilde{\mathbf{u}}\|^2$ is precisely the kinetic energy of the curl-free part, $\frac{1}{2}\|\mathbf{v}\|^2$. Furthermore, the curl-free part $\mathbf{v}$ is determined by the divergence of $\tilde{\mathbf{u}}$ and depends linearly on it; that is, we can write $\mathbf{v} = H(\nabla \cdot \tilde{\mathbf{u}})$ for some linear operator H.[1] If $\nabla \cdot \tilde{\mathbf{u}} = \Theta(\Delta t^k)$, it can be expressed as $\nabla \cdot \tilde{\mathbf{u}} = \delta\, \Delta t^k + o(\Delta t^k)$ for some scalar field δ. Thus we have $\frac{1}{2}\|\mathbf{v}\|^2 = \frac{1}{2}\|H\delta\|^2 \Delta t^{2k} + o(\Delta t^{2k}) = \Theta(\Delta t^{2k})$. □

Using this result, we show that an advection-projection solver can at best only preserve energy to first order in time.

Theorem 2.3.2. *The kinetic energy loss due to the projection step of the advection-projection method is $\Theta(\Delta t^2)$.*

Proof. Expanding the advection operator into its Taylor series,

$$\tilde{\mathbf{u}}^1 = \mathbf{u}^0 - (\mathbf{u}^0 \cdot \nabla)\mathbf{u}^0 \Delta t + O(\Delta t^2). \tag{2.3.3}$$

1. In particular, $H = \nabla \Delta^{-1}$, where Δ^{-1} denotes the inverse of the Laplace operator with the appropriate problem-defined boundary conditions.

we find that the divergence of the advected velocity field is

$$\nabla \cdot \tilde{\mathbf{u}}^1 = \nabla \cdot \mathbf{u}^0 - \nabla \cdot ((\mathbf{u}^0 \cdot \nabla)\mathbf{u}^0)\Delta t + O(\Delta t^2) \tag{2.3.4}$$

$$= \underbrace{-\nabla \cdot ((\mathbf{u}^0 \cdot \nabla)\mathbf{u}^0)}_{\delta(\mathbf{u}^0)} \Delta t + O(\Delta t^2). \tag{2.3.5}$$

For a divergence-free velocity field $\mathbf{u}$, it can be shown that the rate of divergence $\delta(\mathbf{u})$ is proportional to the second invariant of the velocity gradient, $\sum_{ij} \frac{\partial u_i}{\partial x_j}\frac{\partial u_j}{\partial x_i}$, and is not in general zero. Therefore, we have $\nabla \cdot \tilde{\mathbf{u}}^1 = \Theta(\Delta t)$, resulting in an energy loss of order $\Theta(\Delta t^2)$. $\square$

2.3.2. Reflection Solver

We would like to avoid the second-order energy loss during projection while still achieving zero divergence at the end of the time step. The intuition for our approach is that, instead of correcting the velocity at the end of the time step, we can *over-compensate* in the middle: we advect to the middle of the time interval and apply twice the correction needed to obtain a divergence-free field such as to anticipate the divergence incurred during the second half. Geometrically, this operation can be interpreted as symmetrically changing from one side of the divergence-free manifold to the other, hence the term *reflection*; see Fig. 2.2. Concretely, the reflection solver proceeds as follows:

$$\begin{aligned}
\tilde{\mathbf{u}}^{1/2} &= A(\mathbf{u}^0; \mathbf{u}^0, \tfrac{1}{2}\Delta t) \\
\mathbf{u}^{1/2} &= P\tilde{\mathbf{u}}^{1/2} \\
\hat{\mathbf{u}}^{1/2} &= 2\mathbf{u}^{1/2} - \tilde{\mathbf{u}}^{1/2} \\
\tilde{\mathbf{u}}^{1} &= A(\hat{\mathbf{u}}^{1/2}; \mathbf{u}^{1/2}, \tfrac{1}{2}\Delta t) \\
\mathbf{u}^{1} &= P\tilde{\mathbf{u}}^{1} \ .
\end{aligned}$$

As it is preferable to perform semi-Lagrangian advection using a divergence-free velocity field (otherwise the advection equation does not correspond to a conservation law for the advected quantity), we use the *projected* mid-step velocity $\mathbf{u}^{1/2}$ in the second semi-Lagrangian advection step. Note that the reflection velocity $\hat{\mathbf{u}}^{1/2}$ can also be written directly as $\hat{\mathbf{u}}^{1/2} = R\tilde{\mathbf{u}}^{1/2}$, where $R = 2P - \mathbf{I}$ is the reflection operator. The final projection $\mathbf{u}^1 = P\tilde{\mathbf{u}}^1$ guarantees divergence-free velocity at the end of the time step. However, if the rate of divergence does not vary much across the time step, then $\tilde{\mathbf{u}}^1$ will already be close to divergence-free. Indeed, this intuition is confirmed by the following statement.

Theorem 2.3.3. *The kinetic energy loss due to the projection step of the reflection solver is* $O(\Delta t^4)$.

Proof. Using the Taylor series expansion of the advection operator, we approximate the mid-step velocities as

$$\tilde{\mathbf{u}}^{1/2} = \mathbf{u}^0 - (\mathbf{u}^0 \cdot \nabla)\mathbf{u}^0 \Delta t/2 + O(\Delta t^2) \ , \tag{2.3.6}$$

$$\hat{\mathbf{u}}^{1/2} = \mathbf{u}^0 - R(\mathbf{u}^0 \cdot \nabla)\mathbf{u}^0 \Delta t/2 + O(\Delta t^2) \ , \tag{2.3.7}$$

where we have used the fact that $R\mathbf{u}^0 = \mathbf{u}^0$ since $\mathbf{u}^0$ is already divergence-free. An analogous first-order approximation of the end-of-step velocity *before* projection, $\tilde{\mathbf{u}}^1$, yields

$$\tilde{\mathbf{u}}^1 = \hat{\mathbf{u}}^{1/2} - (\mathbf{u}^{1/2} \cdot \nabla)\hat{\mathbf{u}}^{1/2} \Delta t/2 + O(\Delta t^2) \tag{2.3.8}$$

$$= \Big(\mathbf{u}^0 - R(\mathbf{u}^0 \cdot \nabla)\mathbf{u}^0 \Delta t/2\Big) \tag{2.3.9}$$

$$- \Big((\mathbf{u}^0 + O(\Delta t)) \cdot \nabla\Big)\Big(\mathbf{u}^0 + O(\Delta t)\Big)\Delta t/2 + O(\Delta t^2) \tag{2.3.10}$$

$$= \mathbf{u}^0 - R(\mathbf{u}^0 \cdot \nabla)\mathbf{u}^0 \Delta t/2 - (\mathbf{u}^0 \cdot \nabla)\mathbf{u}^0 \Delta t/2 + O(\Delta t^2) \tag{2.3.11}$$

$$= \mathbf{u}^0 - 2P\Big((\mathbf{u}^0 \cdot \nabla)\mathbf{u}^0\Big)\Delta t/2 + O(\Delta t^2) \ . \tag{2.3.12}$$

From this, it is evident that $\tilde{\mathbf{u}}^1$ is divergence-free to first order, i.e. $\nabla \cdot \tilde{\mathbf{u}}^1 = O(\Delta t^2)$. Therefore, the resulting energy loss is of order $O(\Delta t^4)$. □

The good energy-preserving properties of our reflection solver are further assured by the fact that

Theorem 2.3.4. *The reflection operator preserves kinetic energy.*

Proof. The pressure projection $\vec{w} = P\mathbf{u}$ can be interpreted as finding $\vec{w}$ as the closest point to $\mathbf{u}$ in the space of divergence-free vector fields, under the metric defined by kinetic energy [Batty et al., 2007]. Consequently, P is an *orthogonal* projection with respect to kinetic energy, and $\mathbf{v}$ and $\vec{w}$ are orthogonal to each other in the sense that $\langle \mathbf{v}, \vec{w} \rangle = \iiint \rho \mathbf{v} \cdot \vec{w} \, \mathrm{d}V = 0$. Using the definition of the reflection operator, we have for the reflected velocity field $R\mathbf{u} = 2\vec{w} - \mathbf{u} = \vec{w} - \mathbf{v}$. Comparing the kinetic energies, we find that

$$\frac{1}{2}\langle R\mathbf{u}, R\mathbf{u} \rangle = \frac{1}{2}(\langle \vec{w}, \vec{w} \rangle + \langle \mathbf{v}, \mathbf{v} \rangle) = \frac{1}{2}\langle \mathbf{u}, \mathbf{u} \rangle. \tag{2.3.13}$$

□

These results also point to the stability of our method, since the reflection operator preserves energy, while the final projection can only reduce it. In conjuction with the stability of semi-Lagrangian advection operations, one would like to argue that the reflection method is unconditionally stable. This argument does not go through smoothly, however, as advection is stable in a slightly different sense: it is monotonic in the field values, not necessarily in energy. However, the same caveat applies to all other advection-projection solvers used in graphics. As such we expect to see unconditional stability in practice, independent of the time step size.

It is worth noting here that the reflection solver does not necessarily improve the *accuracy* of the solution over projection solvers, since it still incurs error due to self-advection using a "frozen" velocity field. For time-varying flows the method remains only first-order accurate in time, as we show in Section 2.4.3. Instead, the benefit of the reflection approach is a higher degree of energy conservation. This is analogous to how BDF2 and implicit midpoint are both second-order accurate integration schemes, but implicit midpoint has significantly better conservation properties.

2.3.3. Symmetric Projection Methods

The results of the previous section suggest that our reflection solver is *first-order structurally symmetric*: even though the advection operator itself is not, the structure of the advection-reflection-advection sequence is symmetric. This observation prompted us to draw the analogy to symmetric manifold projection methods, a class of integrators for conservative mechanical systems known for their excellent long-term energy conservation [Hairer et al., 2006]. Indeed, by composing the advection-projection step with its adjoint—and vice-versa—we obtain two immediate candidates for symmetric advection-projection schemes.

We start by defining some key terms, following Hairer et al. [2006]. The *adjoint* of a time-stepping scheme $\Phi(\,\cdot\,,h)$ is the inverse of its time reversal, $\Phi^*(\,\cdot\,,h) = \Phi^{-1}(\,\cdot\,,-h)$. A time-stepping scheme is *symmetric*, or *time-reversible*, if it is equal to its adjoint. Composing any consistent first-order scheme Φ with its adjoint yields a symmetric second-order scheme, $\Psi(\,\cdot\,,h) = \Phi^*(\,\cdot\,,h/2) \circ \Phi(\,\cdot\,,h/2)$.

Strictly speaking, the adjoint of the pressure step does not exist because P as a linear operator is not invertible. In this section, we interpret P as an arbitrary perturbation normal to the divergence-free manifold, defining $P(\mathbf{u};p) = \mathbf{u} - \nabla p$ for any pressure field p. This operation is self-adjoint: $P^*(\mathbf{u};p) = P^{-1}(\mathbf{u};-p) = P(\mathbf{u};p)$.

APA^*. The first scheme is obtained by composing the advection-projection step, AP, with its adjoint PA^*. Geometrically, we first advect to the middle of the time step to obtain $\tilde{\mathbf{u}}^{1/2}$ and project onto the divergence-zero manifold. To implement PA^*, we solve for a pressure field p used to compute velocity perturbations $\Delta\mathbf{u}_p = -\nabla p$ normal to the manifold, as well as divergence-free end-of-time-step velocities $\mathbf{u}^1$ such that, when advecting $\mathbf{u}^1$ backward in time, we end up at the perturbed mid-step velocities $\hat{\mathbf{u}}^{1/2} = \tilde{\mathbf{u}}^{1/2} - \nabla p$. This sequence of coupled operations translates into a system of nonlinear equations,

$$\left\{ \begin{array}{c} A(\mathbf{u}^1;\mathbf{u}^1,-\tfrac{1}{2}\Delta t) - (A(\mathbf{u}^0;\mathbf{u}^0,\tfrac{1}{2}\Delta t) - \nabla p) = 0 \\ \nabla\cdot\mathbf{u}^1 = 0 \end{array} \right\}. \tag{2.3.14}$$

in which the projection and perturbation steps combine into a single operation, which is why we mnemonically refer to this scheme as APA^*. We note that this system is similar to the one derived by Mullen et al. [2009]. We solve (2.3.14) with Newton's method and, as do

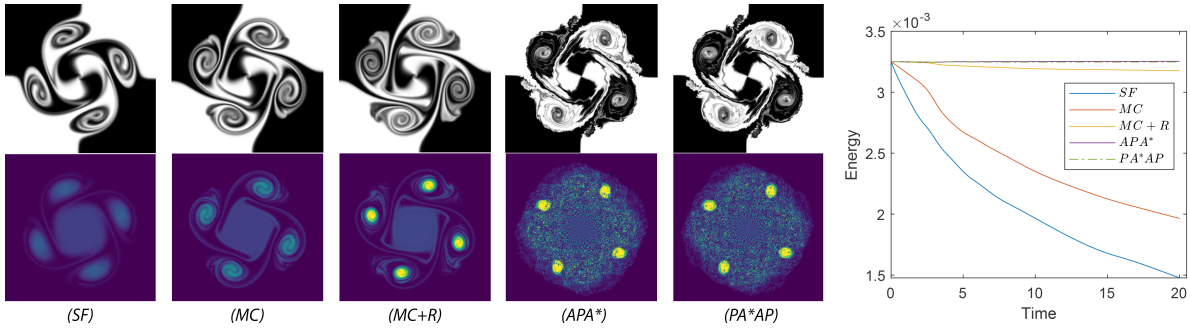


Figure 2.3 – 2D vortex sheet example: comparison of the density distribution (*top*) and the vorticity magnitude (*bottom*) for a fixed point in time ($10s$) as obtained for the various solvers. *Far right*: kinetic energy as a function of time.

Mullen et al., approximate the Jacobian of the advection operator with the identity matrix. When applying block Gaussian elimination to the resulting saddle-point systems, we recover the same Poisson problem encountered in standard advection-projection methods,

$$\nabla^2(\Delta p) = \nabla \cdot \mathbf{r}_{APA^*} \ , \tag{2.3.15}$$

albeit with a different right hand side $\mathbf{r}_{APA^*}$ that we omit here for brevity. This pressure solve must be performed once per Newton iteration, providing us with updated perturbations Δp and corresponding velocity updates.

PA^*AP**.** The second scheme is obtained by composing the adjoint of the advection-projection step with itself, leading to the mnemonic sequence PA^*AP. As a geometric interpretation, we solve for perturbations p and mid-step velocities $\mathbf{u}^{1/2}$ such that 1) advecting from the middle to the end and applying the correction $-\nabla p$ leads to divergence-free velocities, and 2) advecting from the middle backwards leads to the perturbed initial velocities $\mathbf{u}^0 - \nabla p$. Combining these operations together leads to a system of nonlinear equations,

$$\begin{cases} A(\mathbf{u}^{1/2}; \mathbf{u}^{1/2}, -\frac{1}{2}\Delta t) - (\mathbf{u}^0 - \nabla p) = 0 \\ \nabla \cdot \Big(A(\mathbf{u}^{1/2}; \mathbf{u}^{1/2}, \frac{1}{2}\Delta t) - \nabla p\Big) = 0 \end{cases} \tag{2.3.16}$$

that we solve for the mid-step velocities $\mathbf{u}^{1/2}$ and the unknown perturbation field p using Newton's method. This scheme is analogous to the symmetric projection method describe by Hairer et al. [2006] and, upon approximation of the Jacobian with the identity matrix and block Gaussian elimination, leads again to a standard pressure solve (the details of which we leave out for conciseness).

Discussion. We provide qualitative and quantitative evaluations for these symmetric projection methods in Section 2.4 and compare them to our reflection solver. But even before further analysis, we can already expect these methods to have much higher computational cost than our reflection solver: solving the systems to sufficient accuracy requires several Newton iterations, each with one (resp. two) advection steps and a pressure solve. Furthermore, while

the use of an approximate Jacobian is justified by the difficulty of computing the full Hessian, it warrants the use of line search for convergence control. However, since the systems of equations do not derive from a minimization problem with associated objective function, we can only monitor the norm of the residual—a poor measure of progress that can even prevent convergence.

2.4. Results

We investigated the qualitative and quantitative behavior of our reflection solver on a set of 2D and 3D experiments commonly used in the literature. We compare our results to those obtained for conventional advection-projection solvers and report our findings below.

Solvers & Setup. For the 2D experiments, we use our own solver based on a MAC grid discretization. To implement internal obstacles in the flow, we use the method of Batty et al. [2007] and apply corresponding modifications to the matrix of the pressure solves. The 3D examples are based on the Mantaflow library [Thuerey and Pfaff, 2016], which we modified slightly to implement our reflection solver.

We compare the following solvers: stable fluids with first-order semi-Lagrangian advection (*SF*) as described by Stam [1999], stable fluids with MacCormack advection (*MC*) [Selle et al., 2008], our reflection solver (*R*), as well as the symmetric projection methods (APA^*) and (PA^*AP). Statistics on all experiments and solvers—including step size, grid size, and computation time—are listed in Table 2.1. Since our reflection solver requires two advection operations and two pressure solves per time step, we use twice the step size when comparing to *SF* and *MC*, giving us essentially identical computation time.

Table 2.1 – (1) 2D Vortex Sheet, (2) Taylor Vortices, (3) Vortex Shedding, (4) Spiral Maze, (5) Vortex Leap-Frogging, (6) Ink Drop, (7) Smoke Plume, (8) Smoke Plume with Sphere The time step was doubled for the reflection solver when producing the results to keep cost similar to the projection method. The time step was reduced for PA^*AP and APA^* to satisfy the CFL condition.

	Resolution	Domain size	Δt	Iter. time (s) MC	MC+R
1	256×256	1×1	0.025	0.0281	0.055
2	256×256	1×1	0.025	0.0285	0.055
3	512×128	1×0.25	0.0025	0.0322	0.0638
4	256×256	0.75×0.75	0.025	0.0558	0.1107
5	$256 \times 128 \times 128$	$256 \times 128 \times 128$	0.25	5.59	10.43
6	$128 \times 64 \times 64$	$128 \times 64 \times 64$	1	0.27	0.57
7	$128 \times 256 \times 128$	$128 \times 256 \times 128$	1	5.73	10.66
8	$128 \times 256 \times 128$	$128 \times 256 \times 128$	1	7.97	13.48

2.4.1. 2D Results

Since two-dimensional flows are generally easier to interpret and compare, we begin our analysis in the 2D setting.

2D Vortex Sheet. We initialize a disc-shaped region in the center of the scene with a rigid rotation as the initial velocity field. After initialization, no additional energy is injected into the system, allowing us to investigate the (long-term) energy conservation properties of the different solvers.

Fig. 2.3 shows an overview of density and vorticity magnitude fields obtained for the various solvers applied to this problem. While the differences in dynamic behavior are best observed in the accompanying video, it can be seen that our reflection solver *MC+R* shows slightly better detail preservation in the density field than *MC* and much less vorticity diffusion. Moreover, the temporal evolution of kinetic energy shown in Fig. 2.3 (right) speaks a very clear language: *SF* rapidly dissipates energy, which drops to two thirds of its initial value after roughly $6s$. *MC* performs better, but still loses one third of its energy after $13s$. By contrast, our reflection solver preserves energy much better, losing less than 3% over the entire animation ($20s$). For reference, the symmetric projection methods both preserve energy perfectly, but they require roughly 10 times more computation time. What is worse, however, is that both methods lead to visually disturbing artifacts in the density field and very noisy vorticity (see Fig. 2.3). We have investigated this behavior intensively but could not find a problem with our implementation. We conjecture that the reason for these artifacts lies in the semi-Lagrangian advection operator: the interpolations performed when tracing back through the velocity field act as a low-pass filter, i.e., information is lost due to interpolation. Although the continuity of the flow provides some amount of regularization, the effective inversion of this low-pass filter during A^* is numerically unstable, explaining both the artifacts and the intermittent convergence problems that we observed. Although we intended these symmetric methods to be reference solutions, we found them to be unusable in practice and will therefore not discuss them further.

Taylor Vortices. Another frequently used 2D example is that of two vortices of the same sign placed at a given initial distance. Depending on this initial distance, the analytical solution for this problem will lead to the vortices either merging or separating. Following Mullen et al. [2009], we choose the initial distance slightly larger than the critical value and investigate the behavior of the solvers. For both *SF* and *MC*, the vortices merge shortly after the beginning of the simulation, whereas they separate as predicted by the analytical solution using our reflection solver.

Vortex Shedding. This example investigates the behavior of the different solvers in combination with internal boundaries, leading to dynamic and visually rich vortex shedding. As can be seen from Fig. 2.5, *MC* initially produces similar behavior in the sense that

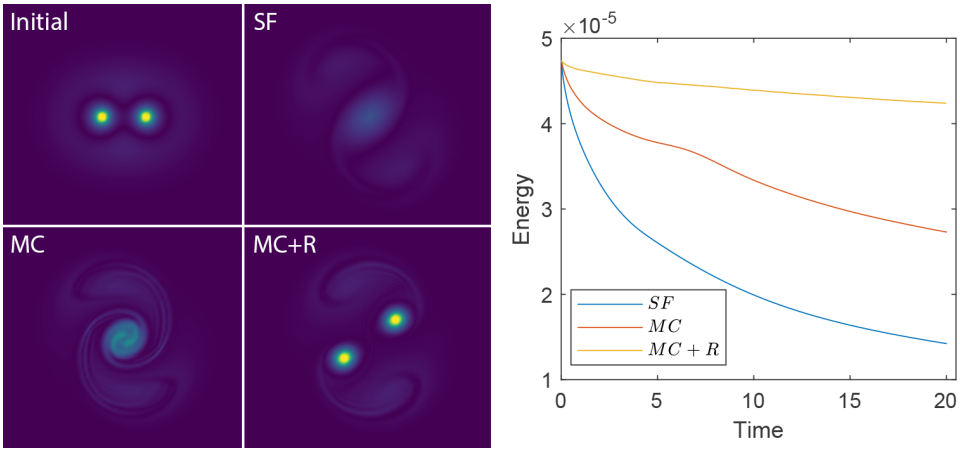


Figure 2.4 – 2D Taylor vortices. *Left*: initial vorticity magnitude and results for the three solvers after $10s$. *Right*: kinetic energy as a function of time.

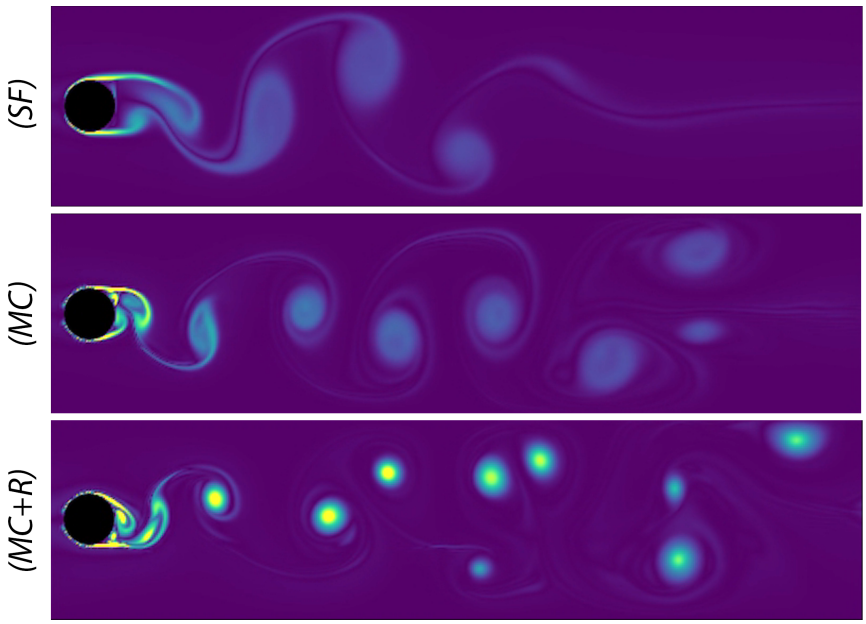


Figure 2.5 – 2D vortex shedding: comparison of the vorticity magnitude distributions for *SF*, *MC*, and *MC+R* for a fixed point in time ($6s$).

roughly the same number of vortices is shed during the first 6 seconds. However, whereas our reflection solver approximately preserves the vorticity of the shed vortices, there is a clear decay in vorticity (from left to right) for *MC*. The video also shows that the behavior for *SF* is qualitatively very different and the numerical viscosity is clearly visible.

Spiral Maze. An example introduced by Mullen et al. [2009] contains a pair of vortices in a 2D domain with many boundaries forming a spiral maze. In the absence of dissipation, one of the vortices should advect itself to the center of the maze. As shown in Figure 2.6, our method produces the expected behavior. Interestingly, unlike the results of Mullen et al., we

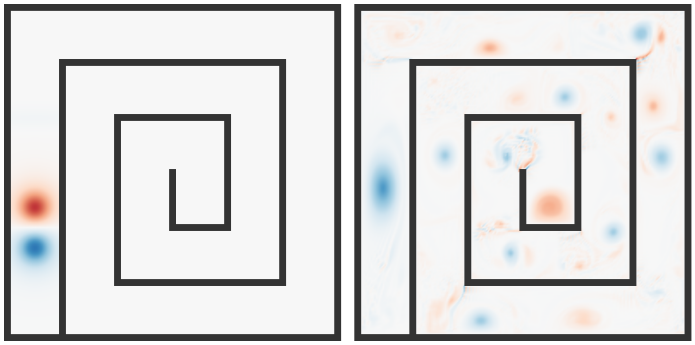


Figure 2.6 – A vortex in a spiral maze (*left*) correctly advects itself to the center (*right*).

also observe additional vortex shedding due to flow separation at the convex corners of the maze.

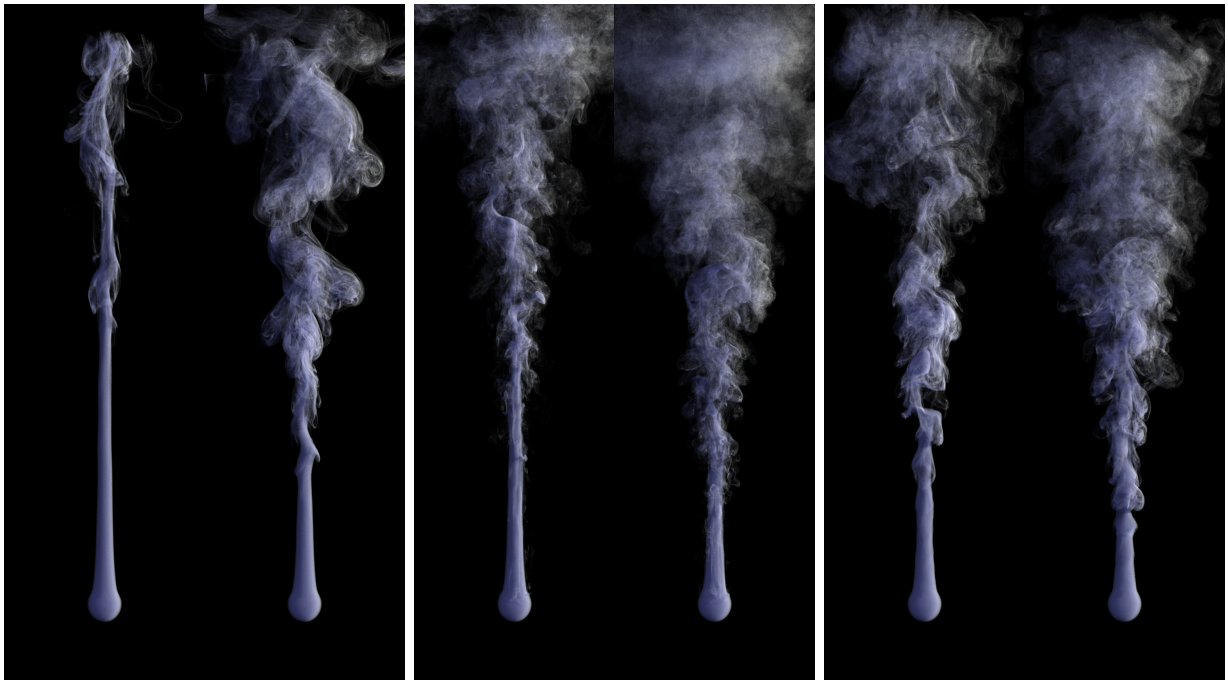
2.4.2. 3D Results

To investigate the behavior of our reflection solver in 3D, we chose a set of examples frequently used in the literature.

Vortex Leap-Frogging. This classical example is initialized with two concentric vortex rings of different radii but equal circulations. In the analytical solution for the completely inviscid, conservative case, the two vortices will produce a *leap-frogging* motion that continues indefinitely. However, reproducing this behavior with Eulerian methods has been a challenging problem due to the tendency for numerical dissipation to diffuse the vortices into each other. As can be seen in Fig. 2.1, when using the *MC* solver, the vortex rings merge into a single one during the first leap-through motion. By contrast, our solver successfully produces several leap-frogging moves in which the rings remain clearly separated. We refer to the accompanying video for a better illustration of this fascinating phenomenon.

Ink Drop. A spherical density field is initialized with constant horizontal velocity. Using our reflection solver, the velocity gradient at the sphere's interface immediately leads to the formation of a large vortex, leaving behind it a trail of turbulent fluid that develops into increasingly complex patterns. While the *MC* solver can reproduce the vortex formation, the flow has much more viscosity and the trail it leaves behind is devoid of any detail. Please refer to the accompanying video.

Smoke Plume. As another classical example, we simulate a smoke plume by modeling a spherical density source subject to a constant density-proportional buoyancy. The action of buoyancy leads to the formation of a characteristic vortex front, followed by turbulent breakup of the rising smoke column; see Fig. 2.7. We have compared the standard advection-projection scheme with our advection-reflection solver with three different advection schemes: *SF*, *MC*, and *FLIP* [Zhu and Bridson, 2005]. While the overall qualitative behavior is similar for most cases, our reflection solver uniformly yields more pronounced vortical motion and stronger



SF *MC* *FLIP*

Figure 2.7 – 3D smoke plume with different advection schemes. In each pair, we compare the same simulation frame computed using a standard projection solver (*left*) and our reflection solver (*right*).

turbulence due to the reduced artificial dissipation. In the accompanying video, we also show an example with a spherical obstacle in the path of the plume.

2.4.3. Convergence Analysis

In order to validate our theoretical result for the improved order of accuracy of our method, we compared different solvers with different parameters on a 2D smoke plume example. We let the simulation run for $2s$ such that an interesting flow field develops, and compute the divergence before projection at the end of a given time step. Fig. 2.8 shows a log-log plot of this pre-projection divergence as a function of the step size. It can be seen that, for the *SF* and *MC* methods, the divergence decreases linearly with the step size. For our reflection solver, however, the decrease is indeed quadratic, irrespective of whether it is combined with first order or MacCormack advection. Another observation that we made in this context is that because in this test we only increased the resolution in time, not in space, clamping the interpolated field value in the MacCormack scheme slows the convergence of the norm of the divergence.

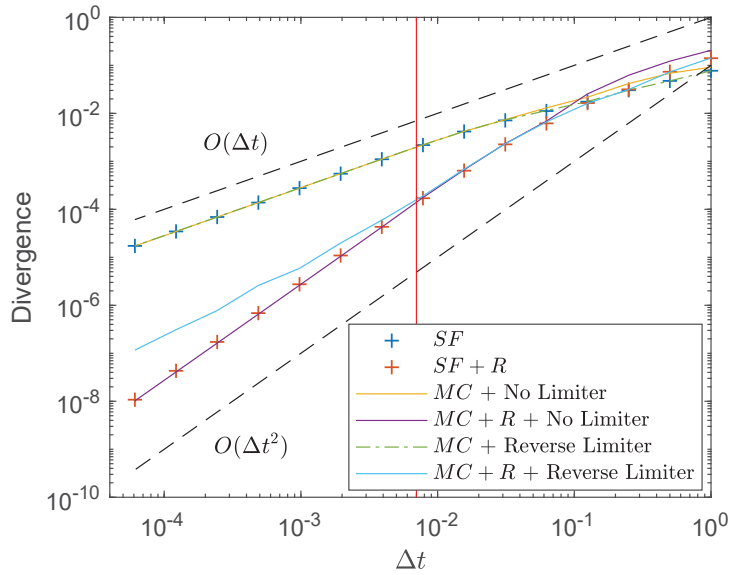


Figure 2.8 – Norm of the divergence before the projection step for different step sizes and different solvers. *Reverse Limiter* refers to the clamping of the interpolated field value in the MacCormack scheme.

To evaluate the accuracy of our method, we performed a convergence test using a 2D Taylor-Green vortex with initial conditions

$$\mathbf{u}_{\mathrm{TG}} = (\sin(2\pi x)\cos(2\pi y), -\cos(2\pi x)\sin(2\pi y)).$$

In the inviscid case, this is a steady flow with $\mathbf{u}$ constant over time. We also simulated an example with initial velocity $\mathbf{u} = \mathbf{u}_{\mathrm{TG}} + (1,0)$ and periodic boundary conditions, which should

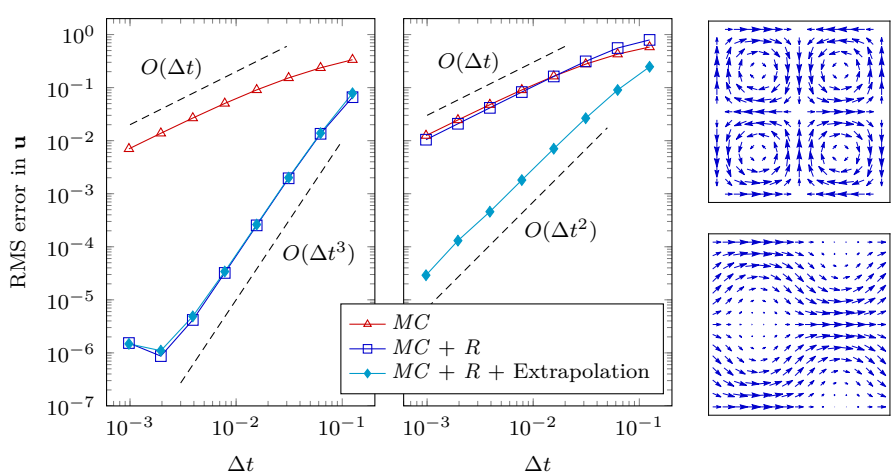


Figure 2.9 – RMS error in velocity after one second for (*left*) the 2D Taylor-Green vortex, and (*middle*) the Taylor-Green vortex with added translation. "Extrapolation" denotes advection with the velocity field $2\mathbf{u}^{1/2} - \mathbf{u}^0$ in the second half-step. *Right*: Initial velocity fields for both cases are visualized.

result in a pure translation of the vortex. We ran both examples to 1s, and computed the RMS error in velocity with respect to the analytical solution. Figure 2.9 shows a log-log plot of this error as a function of time step. For the steady flow, our reflection solver exhibits third-order convergence, since energy loss is the only source of error in this case. For the unsteady flow, the error caused by the use of the "frozen" velocity field in self-advection dominates, and we observe similar first-order convergence for both projection and reflection methods. However, when performing this test we noticed that convergence appears to be improved to second order by performing the second advection half-step using an "extrapolated" velocity field $2\mathbf{u}^{1/2} - \mathbf{u}^0$, as a first-order approximation of $\mathbf{u}^1$. We leave a detailed investigation of this effect to future work.

2.5. Conclusions

We presented a new method for detail-preserving fluid animation that improves on current advection-projection solver: it replaces the energy-dissipating projection applied at the end of a simulation step with an energy-preserving reflection in the middle of the interval.

Compared to existing advection-projection methods, the central advantages of our reflection solver are that *i*) it has provably better energy-conservation properties, and *ii*) it empirically leads to much less vorticity diffusion and preserves more detail for equal computational costs. Furthermore, our method integrates readily with existing grid-based solvers and is very easy to implement.

2.5.1. Limitations & Future Work

While our reflection solver preserves energy and vorticity very well, it does not do so perfectly. One reason is that the semi-Lagrangian advection step is itself not energy-preserving. Another one is that the projection at the end still removes energy from the system, albeit at a much smaller rate than current advection-projection solvers.

While the improved energy and vorticity behavior generally lead to visually richer animations, our solver does not introduce detail where none should arise according to the true solution. In visual effects, however, it is often desirable to *artificially enhance* results, e.g., by amplifying vorticity or injecting turbulence. Our solver can, in principle, be used on conjunction with such methods and it would be interesting to investigate its behavior in this context. Furthermore, we also like to combine our reflection solver with the recent *advection-enhancing* methods of Zhang et al. [2015] and Chern et al. [2016].

Extensions to simplicial grids are another direction worth exploring. Finally, we would like to apply the reflection solver to other constraint-projection applications such as inextensible cloth [Goldenthal et al., 2007] and volume-preserving solids [Irving et al., 2007] that have so far relied on the step-and-project approach.

Acknowledgments

We thank the anonymous reviewers for their constructive and thoughtful feedback. This work was supported by the Discovery Grants Program of the Natural Sciences and Engineering Research Council of Canada (NSERC).

Chapter 3

SGN: Sparse Gauss-Newton for Accelerated Sensitivity Analysis

by

Jonas Zehnder[1], Stelian Coros[2], and Bernhard Thomaszewski[1,2]

([1]) Université de Montréal

([2]) ETH Zürich

Publication

The article has been submitted for publication in ACM Transactions on Graphics. It has been accepted for publication and is expected to appear in Volume 41, Issue 1 (February 2022). The article was reformatted to fit the style of this thesis.

Jonas Zehnder contributed the numerical experiments and wrote a draft of the paper. Bernhard Thomaszewski and Stelian Coros were supervising the whole process and editing the paper.

Abstract. We present a sparse Gauss-Newton solver for accelerated sensitivity analysis with applications to a wide range of equilibrium-constrained optimization problems. Dense Gauss-Newton solvers have shown promising convergence rates for inverse problems, but the cost of assembling and factorizing the associated matrices has so far been a major stumbling block. In this work, we show how the dense Gauss-Newton Hessian can be transformed into an equivalent sparse matrix that can be assembled and factorized much more efficiently. This leads to drastically reduced computation times for many inverse problems, which we demonstrate on a diverse set of examples. We furthermore show links between sensitivity analysis and nonlinear programming approaches based on Lagrange multipliers and prove equivalence under specific assumptions that apply for our problem setting.
Keywords: sensitivity analysis, Gauss-Newton

3.1. Introduction

Many design tasks in engineering involve the solution of *inverse problems*, where the goal is to find design parameters for a mechanical system such that the corresponding equilibrium state is optimal with respect to given objectives. As an alternative to conventional nonlinear programming, an approach that has recently seen increasing attention in the visual computing community is to eliminate the equilibrium constraints using sensitivity analysis. Removing redundant degrees of freedom decreases not only the problem size, it also transforms a difficult nonlinear constrained optimization problem into an unconstrained minimization problem.

Solving such minimization problems efficiently requires derivatives of the map between parameters and state, which is given implicitly via the solution of the *forward* simulation problem. While the gradient can be computed efficiently with the adjoint method, using only first-order derivative information often leads to unsatisfying convergence, even if acceleration techniques such as L-BFGS are used. Second-order sensitivity analysis promises faster convergence by requiring fewer iterations but comes at the price of dense and potentially indefinite system matrices. The second problem can be resolved by resorting to the Gauss-Newton method, which replaces the full Hessian with a positive-definite approximation. However, its dense nature greatly impedes the potential of second-order sensitivity analysis.

In this paper, we show how the dense Gauss-Newton Hessian can be transformed into an equivalent sparse matrix that can be assembled and factorized much more efficiently than its dense counterpart. Whereas the asymptotic complexity for dense solvers is approximately $O(n^3)$ for an $n \times n$ matrix, the cost of factorizing sparse systems depends on the sparsity pattern, which is itself problem-dependent. While meaningful asymptotic bounds for sparse factorization are hard to obtain [Peng and Vempala, 2020], our extensive numerical examples indicate drastically reduced computation times for a wide range of inverse design problems. We furthermore establish links between sensitivity analysis and general nonlinear programming approaches based on Lagrange multipliers and prove equivalence under specific assumptions.

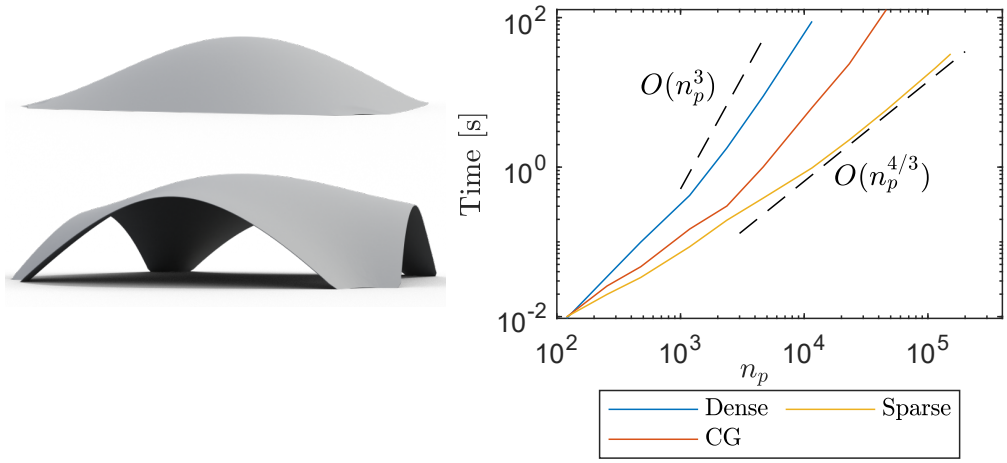


Figure 3.1 – Comparison between dense Gauss-Newton and our sparse formulation on a shape optimization problem for a concrete shell. *Left*: shell roof before (*top*) and after (*bottom*) optimization. *Right*: average timings for computing the search direction with sparse Gauss-Newton, dense Gauss-Newton and the CG method as a function of the number of design parameters n_p.

3.2. Related Work

Since fabrication-oriented design moved into the focus of the visual computing community, inverse problems have received a surge of attention. While an exhaustive review of applications is beyond the scope of this work, many different methods have been proposed for solving inverse design problems ranging from musical instruments [Bharaj et al., 2015; Musialski et al., 2016; Umetani et al., 2016] and balloons [Skouras et al., 2014, 2012], to deployable structures [Guseinov et al., 2017; Kilian et al., 2017; Panetta et al., 2019] and architectural-scale surfaces [Vouga et al., 2012] and frameworks [Jiang et al., 2017; Gauge et al., 2014; Pietroni et al., 2017]. While some of these works propose formulations tailored to specific applications, we target general inverse problems that can be cast as constrained minimization problems with continuous parameter and state variables that are subject to equality constraints derived from physical principles.

One natural approach to solving such problems is via sequential quadratic programming (SQP), which uses states and parameters as problem variables while introducing Lagrange multipliers to enforce equilibrium constraints; see, e.g., [Skouras et al., 2014]. As an alternative that avoids challenges associated with Lagrange multiplier formulations, the augmented Lagrangian method (ALM) has been applied to shape [Skouras et al., 2012] and multi-material [Skouras et al., 2013] optimization problems. Another approach that has recently seen increasing attention is sensitivity analysis, which eliminates state variables and constraints such as to obtain an unconstrained minimization problem with design parameters as only variables. Sensitivity analysis is a powerful method that has been used for inverse design of mechanisms [Coros et al., 2013; Megaro et al., 2017], clothing [Wang, 2018], material

optimization [Yan et al., 2018; Zehnder et al., 2017] as well as for optimization-based forward design [Umetani et al., 2011; Pérez et al., 2017].

First-order sensitivity analysis provides the gradient of the objective function with respect to design parameters, which can be computed efficiently using the adjoint method (see, e.g., [Bletzinger et al., 2010]). While gradient descent typically performs poorly, faster convergence for the corresponding minimization problem can be achieved using, e.g., Anderson acceleration [Peng et al., 2018], or Quasi-Newton methods such as L-BFGS [Liu et al., 2017]. Kovalsky et al. [2016] improve convergence for mesh optimization by combining acceleration with Laplacian preconditioning for the gradient. Using the same preconditioner, Zhu et al. [2018] instead propose a modified L-BFGS method to accelerate convergence. These methods mostly aim at geometry deformation tasks for which the cost of evaluating the objective is relatively low. For physics-constrained design problems, however, the costs per step are significantly higher since each evaluation of the objective function requires simulation.

Recent work has started to investigate ways of exploiting second-order derivative information for sensitivity analysis. Panetta et al. [2019] use Newton's method with a trust region approach on the reduced Hessian resulting from second-order sensitivity analysis. Zimmermann et al. [2019] propose a Generalized Gauss-Newton solver with Hessian contributions selected such as to avoid indefinite matrices. However, while this method has led to promising convergence in terms of the number of required iterations, assembling and factorizing the dense Hessian undoes this potential advantage to a large extent. This impression is further substantiated by Wang [2018], who benchmarked sensitivity analysis with generalized Gauss-Newton against exact and inexact gradient descent methods.

Our method overcomes these problems through a sparse reformulation of Gauss-Newton that yields the same search direction but drastically decreases the time required for assembling and factorizing the linear system. Although sparse Gauss-Newton formulations for sensitivity analysis have, to the best of our knowledge, not been investigated before, there are connections to so-called projected SQP methods based on reduced Hessians that have been studied in the optimization community [Heinkenschloss, 1996]. We use these insights to show equivalence between sensitivity analysis and nonlinear programming with Lagrange multipliers for specific equilibrium-constrained optimization problems.

3.3. Background

We consider constrained optimization problems of the form

$$\min_{\mathbf{x},\mathbf{p}} f(\mathbf{x},\mathbf{p}) \quad \text{s.t.} \quad \mathbf{c}(\mathbf{x},\mathbf{p}) = \mathbf{0} \ , \tag{3.3.1}$$

where $\mathbf{x}$ denotes the equilibrium state of a mechanical system described by a set of design parameters $\mathbf{p}$. The state $\mathbf{x}$ is coupled to the design parameters $\mathbf{p}$ through a set of constraints

$\mathbf{c}$ requiring that $\mathbf{x}$ must be an equilibrium configuration for $\mathbf{p}$. While the exact form of these equilibrium constraints depends on the problem, we will focus on static and dynamic force balance in this work.

3.3.1. Sensitivity Analysis

We exclusively consider the special but common case of problems which exhibit exactly as many equality constraints as state variables, i.e., $n_{\mathbf{c}} = n_{\mathbf{x}}$. Furthermore, we assume that the constraint Jacobian $\frac{\partial \mathbf{c}}{\partial \mathbf{x}}$ has full rank. Under these conditions, the implicit function theorem asserts that any choice of $\mathbf{p}$ in a local neighborhood uniquely determines the corresponding equilibrium state $\mathbf{x}$ and we therefore write $\mathbf{x} = \mathbf{x}(\mathbf{p})$. Given a state-parameter pair $(\mathbf{x},\mathbf{p})$ that satisfies the equilibrium constraints, we require that any change to the design parameters induces a corresponding change in state such that the system is again at equilibrium. Formally, we have

$$\frac{d\mathbf{c}}{d\mathbf{p}} = \frac{\partial \mathbf{c}}{\partial \mathbf{p}} + \frac{\partial \mathbf{c}}{\partial \mathbf{x}} \frac{d\mathbf{x}}{d\mathbf{p}} = \mathbf{0} \ , \tag{3.3.2}$$

from which we directly obtain the so called *sensitivity matrix*

$$\mathbf{S} = \frac{d\mathbf{x}}{d\mathbf{p}} = -\left(\frac{\partial \mathbf{c}}{\partial \mathbf{x}}\right)^{-1} \frac{\partial \mathbf{c}}{\partial \mathbf{p}} \ . \tag{3.3.3}$$

Using this relation, the gradient of the objective function with respect to the parameters is obtained as

$$\frac{df(\mathbf{x}(\mathbf{p}),\mathbf{p})}{d\mathbf{p}} = \frac{\partial f}{\partial \mathbf{p}} + \frac{\partial f}{\partial \mathbf{x}} \mathbf{S} \ . \tag{3.3.4}$$

We note that, by using (3.3.2) in the above expression and rearranging terms, computing the gradient requires only the solution of a single linear system.

3.3.2. Gauss-Newton

With the gradient defined through (3.3.4), we can minimize f using steepest descent in parameter space. Every step amounts to updating $\mathbf{p}$ along the search direction, computing an equilibrium configuration $\mathbf{x}$ through simulation, and evaluating the objective to accept or reject the step. Though simple, the convergence of steepest descent is typically very slow. Using the Hessian of the objective function, Newton's method promises quadratic convergence close to the solution. However, Newton's method is often plagued by indefiniteness on the road towards the optimum, requiring expensive regularization and other advanced strategies. As a promising middle ground, Gauss-Newton retains parts of the Hessian information but is guaranteed to never encounter indefiniteness.

Gauss-Newton in its original form is a minimization algorithm for objective functions in nonlinear least-squares form,

$$f(\mathbf{x}, \mathbf{p}) = \sum_i \frac{w_i}{2} r_i(\mathbf{x}, \mathbf{p})^2 \ , \tag{3.3.5}$$

where $\vec{w} = (w_1, \ldots, w_n)$ is a vector of weights and r_i are residuals. Instead of using the full Hessian

$$H = \sum_i w_i \frac{dr_i}{d\mathbf{p}}^T \frac{dr_i}{d\mathbf{p}} + \sum_i w_i r_i \frac{d^2 r_i}{d\mathbf{p}^2} \ , \tag{3.3.6}$$

Gauss-Newton drops the second term to define an approximate but positive-definite Hessian. Writing out $dr_i/d\mathbf{p}$ we arrive at

$$H_{GN} = \begin{bmatrix} \frac{d\mathbf{x}}{d\mathbf{p}}^T & I \end{bmatrix} \left(\sum_i w_i \begin{bmatrix} \frac{\partial r_i}{\partial \mathbf{x}}^T \frac{\partial r_i}{\partial \mathbf{x}} & \frac{\partial r_i}{\partial \mathbf{x}}^T \frac{\partial r_i}{\partial \mathbf{p}} \\ \frac{\partial r_i}{\partial \mathbf{p}}^T \frac{\partial r_i}{\partial \mathbf{x}} & \frac{\partial r_i}{\partial \mathbf{p}}^T \frac{\partial r_i}{\partial \mathbf{p}} \end{bmatrix} \right) \begin{bmatrix} \frac{d\mathbf{x}}{d\mathbf{p}} \\ I \end{bmatrix} \ . \tag{3.3.7}$$

A Gauss-Newton step can then be computed by solving the system of linear equations

$$H_{GN} \cdot \delta\mathbf{p} = -\frac{df}{d\mathbf{p}}^T \ . \tag{3.3.8}$$

In practice, however, computing, assembling, and factorizing this reduced Hessian matrix is exceedingly expensive: it requires the complete sensitivity matrix, products between sparse matrices with incompatible sparsity patterns, and leads to a dense matrix that is expensive to factorize.

3.4. Sparse Gauss-Newton

To arrive at a more efficient formulation, we start by rewriting the Gauss-Newton Hessian as

$$H_{GN} = \frac{d\mathbf{x}}{d\mathbf{p}}^T A \frac{d\mathbf{x}}{d\mathbf{p}} + B \frac{d\mathbf{x}}{d\mathbf{p}} + \frac{d\mathbf{x}}{d\mathbf{p}}^T B^T + C \tag{3.4.1}$$

with

$$A = \sum_i w_i \frac{\partial r_i}{\partial \mathbf{x}}^T \frac{\partial r_i}{\partial \mathbf{x}} \ , \ B = \sum_i w_i \frac{\partial r_i}{\partial \mathbf{p}}^T \frac{\partial r_i}{\partial \mathbf{x}} \ , \text{ and } \ C = \sum_i w_i \frac{\partial r_i}{\partial \mathbf{p}}^T \frac{\partial r_i}{\partial \mathbf{p}}.$$

Since $\frac{d\mathbf{x}}{d\mathbf{p}} = -\left[\frac{\partial \mathbf{c}}{\partial \mathbf{x}}\right]^{-1} \frac{\partial \mathbf{c}}{\partial \mathbf{p}}$, the inverse of the constraint Jacobian appears inside the definition of H_{GN}. We can remove this inverse by reformulating the problem with additional variables $\delta\mathbf{x} = \frac{d\mathbf{x}}{d\mathbf{p}}\delta\mathbf{p}$, which is equivalent to $\frac{\partial \mathbf{c}}{\partial \mathbf{p}}\delta\mathbf{p} + \frac{\partial \mathbf{c}}{\partial \mathbf{x}}\delta\mathbf{x} = 0$,

$$\begin{bmatrix} \frac{d\mathbf{x}^T}{d\mathbf{p}} A + B & C + \frac{d\mathbf{x}^T}{d\mathbf{p}} B^T \\ \frac{\partial \mathbf{c}}{\partial \mathbf{x}} & \frac{\partial \mathbf{c}}{\partial \mathbf{p}} \end{bmatrix} \begin{bmatrix} \delta\mathbf{x} \\ \delta\mathbf{p} \end{bmatrix} = \begin{bmatrix} -\frac{df}{d\mathbf{p}}^T \\ 0 \end{bmatrix} \ . \tag{3.4.2}$$

However, the transpose of the sensitivity matrix still appears in this system. To also remove this occurrence, we introduce additional variables $\delta\boldsymbol{\lambda}$ defined as

$$\frac{\partial \mathbf{c}}{\partial \mathbf{x}}^T \delta\boldsymbol{\lambda} + B^T \delta\mathbf{p} + A\delta\mathbf{x} = 0 \ , \tag{3.4.3}$$

and arrive at the extended system

$$\begin{bmatrix} A & B^T & \frac{\partial \mathbf{c}}{\partial \mathbf{x}}^T \\ B & C & \frac{\partial \mathbf{c}}{\partial \mathbf{p}}^T \\ \frac{\partial \mathbf{c}}{\partial \mathbf{x}} & \frac{\partial \mathbf{c}}{\partial \mathbf{p}} & 0 \end{bmatrix} \begin{bmatrix} \delta\mathbf{x} \\ \delta\mathbf{p} \\ \delta\boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} 0 \\ -\frac{df}{d\mathbf{p}}^T \\ 0 \end{bmatrix} \ . \tag{3.4.4}$$

Note that the first row enforces (3.4.3), whereas the second row is obtained by using (3.4.3) and (3.3.3) in the first row of (3.4.2). The resulting system is sparse and it requires neither the inverse of the constraint Jacobian, nor the sensitivity matrix. We emphasize that, by construction, the solution $\delta\mathbf{p}$ obtained when solving this system is *exactly identical* to the one obtained when factorizing the dense Hessian. Although this new system is larger than the reduced one, its sparsity allows us to leverage specialized linear solvers. The exact time complexity of common sparse direct solvers is only known for specific sparsity patterns and can range from $O(n)$ for, e.g., a diagonal matrix to $O(n^3)$ for a quasi-dense matrix [Peng and Vempala, 2020]. Nevertheless, our experiments show that the cost of factorizing the larger sparse system is *asymptotically lower* than the cost of factorizing the reduced dense system for many problems; see Fig. 3.1 for an example. This result translates into dramatically improved performance for a large range of problems, as we demonstrate with our examples.

3.4.1. Discussion and Generalization

Relation to Sequential Quadratic Programming. System (3.4.4) is in the form of a saddle-point problem that is characteristic for first-order optimality conditions in nonlinear programming. Indeed, it can be shown that second-order sensitivity analysis on general objectives is equivalent to so called reduced SQP methods when using a particular definition for the Lagrange multipliers; see [De los Reyes, 2015] and our derivations in Appendix 3.A.1.

Generalization to Arbitrary Objectives. Our construction can be extended to general objectives $f(\mathbf{x}(\mathbf{p}),\mathbf{p})$ for which the Hessian reads

$$\begin{aligned} \frac{d^2 f}{d\mathbf{p}^2} =& \frac{d\mathbf{x}}{d\mathbf{p}}^T \frac{\partial^2 f}{\partial \mathbf{x}^2} \frac{d\mathbf{x}}{d\mathbf{p}} + \frac{\partial^2 f}{\partial \mathbf{x} \partial \mathbf{p}} \frac{d\mathbf{x}}{d\mathbf{p}} + \frac{d\mathbf{x}}{d\mathbf{p}}^T \frac{\partial^2 f}{\partial \mathbf{p} \partial \mathbf{x}} + \frac{\partial^2 f}{\partial \mathbf{p}^2} \\ &+ \sum_i \frac{\partial f}{\partial \mathbf{x}_i} \frac{d^2 \mathbf{x}_i}{d\mathbf{p}^2} \ . \end{aligned} \tag{3.4.5}$$

In particular, when dropping only second-order sensitivities to obtain the Generalized Gauss-Newton approximation [Zimmermann et al., 2019], our formulation applies directly with blocks defined as $A = \frac{\partial^2 f}{\partial \mathbf{x}^2}$, $B = \frac{\partial^2 f}{\partial \mathbf{x} \partial \mathbf{p}}$, and $C = \frac{\partial^2 f}{\partial \mathbf{p}^2}$. The extension to the full-Hessian case

and its relation to nonlinear programming is described in Appendix 3.A.1. It should be noted, however, that neither Generalized Gauss-Newton nor full Newton offer any guarantees on the positive-definiteness of the blocks. In our experiments, the additional measures required for detecting and treating indefiniteness can easily undo the potential advantage of using more accurate Hessian information.

Combination with L-BFGS. As we show in Sec. 3.5, Gauss-Newton leads to very good convergence in many cases and our sparse formulation makes this approach highly efficient. Nevertheless, Gauss-Newton is not a true second-order method and, depending on the problem, the missing derivative information can slow down convergence. For such cases, combining Sparse Gauss-Newton with L-BFGS can be an attractive alternative: even though L-BFGS requires only first-order derivatives, it approximates second-order information in its inverse Hessian from rank-one updates with past gradients. Similar in spirit to [Kovalsky et al., 2016; Liu et al., 2017; Zhu et al., 2018], we use Sparse Gauss-Newton to initialize the inverse Hessian approximation in L-BFGS. This amounts to solving a linear system each time a new search direction is computed. We provide an evaluation of this approach in Sec. 3.5.

Block Solve. If the objective f does not directly depend on the design parameters, the block structure for (3.4.4) simplifies to $B = 0$ and $C = 0$. If the constraint Jacobian $\partial\mathbf{c}/\partial\mathbf{p}$ is invertible , upon block substitution, the solution $\delta\mathbf{p}$ is obtained by solving the linear system

$$\frac{\partial \mathbf{c}}{\partial \mathbf{p}}\delta\mathbf{p} = -\frac{\partial \mathbf{c}}{\partial \mathbf{x}}\delta\vec{y}\,, \quad \text{with} \quad \delta\vec{y} = A^{-1}\frac{\partial f}{\partial x}^T \quad . \tag{3.4.6}$$

See Appendix 3.A.2 for a detailed derivation. If the objective is not a simple L_2 distance, then $A \neq I$ and we must solve an additional linear system to obtain $\delta\vec{y}$. We show in Sec. 3.5 that, where applicable, this block solve can accelerate the computation of the search direction by another 30% and more compared to the Sparse Gauss-Newton baseline.

3.5. Results

We evaluate the performance of our Sparse Gauss-Newton (SGN) solver on a set of inverse design problems. Besides illustrating different applications, each of these problems differs in terms of the ratio between parameters and state variables, the connectivity among variables, as well as their degree of nonlinearity and convexity. We are primarily interested in assessing the relative performance of SGN and dense Gauss-Newton (DGN), and how this ratio evolves as a function of problem size. Since both methods give the same results, we only provide average computation times for computing search directions in most cases. Additionally, we also provide total computation times on selected examples and compare to alternative approaches. We measure convergence in terms of suboptimality, which we define as the objective function value minus its value at the minimum.

Solving the saddle point problem. We used the PARDISO LU direct solver from Intel's Math Kernel Library (MKL) for solving the indefinite sparse linear systems. This solver performed robustly and efficiently for all problem types and resolutions except for the cloth example, where it returned solutions of insufficient accuracy. Instead of tweaking solver parameters per problem, we opted for a robust fall-back strategy based on the iterative BiCGSTAB method [Van der Vorst, 1992], using PARDISO's LDL^Tdecomposition of the stabilized matrix as preconditioner. Specifically, we add the vector $[\varepsilon_x \mathbb{1}_{n_x}^T, \mathbb{0}_{n_p}^T, -\varepsilon_\lambda \mathbb{1}_{n_c}^T]^T$ to the diagonal of the matrix with $\varepsilon_x = 10^{-6}$ and $\varepsilon_\lambda = 10^{-6}$. We found this strategy to work well in practice, requiring only a few BiCGSTAB iterations to solve the system to high accuracy. It should be noted that the optimal stabilization of the lower right block is scale dependent and should be chosen according to the norm of the constraint Jacobian. See [Benzi et al., 2005] for more details on stabilization, and on the numerical solution of saddle point problems in general.

We also experimented with iterative solvers such as BiCGSTAB and GMRES with a variety of commonly used preconditioners, ranging from simple diagonal scaling (Jacobi) to incomplete factorization (ILUT) methods. For the problems considered in this work, however, these iterative methods were either much slower or failed to converge at all. Though we expect iterative solvers to eventually outperform direct solvers for increasingly large problems sizes, we consider this topic beyond the scope of this work. For Sparse Newton and Sparse GGN we used the inertia revealing feature of PARDISO's LDL^Tdecomposition to test for positive-definiteness on the nullspace of the constraint Jacobian (i.e., second-order optimality conditions) and added diagonal regularization if necessary [Han and Fujiwara, 1985]. For the trust region method we used *trlib* [Lenders et al., 2018] to solve the trust region subproblem. We solve the reduced linear systems (DGN) using Eigen's built-in Cholesky decomposition. All measurements that we present here were done on an Intel i7-6700K quad-core with 16GB of RAM.

Computing Equilibrium States. All methods based on sensitivity analysis must recompute the equilibrium state through forward simulation after each parameter update. Forward simulation amounts to a nonlinear minimization problem, whose objective function depends on the application. In each case, we ensure monotonicity in the objective using a backtracking line search. We use standard computational models described in the literature. Derivatives with respect to state and design parameters are computed analytically using pre-compile-time automatic differentiation.

3.5.1. Inverse Elastic Design

Our first example considers gravity compensation for a simple elastic bar clamped on one side and subjected to gravity; see Fig. 3.2. The goal is to find a rest state mesh such that

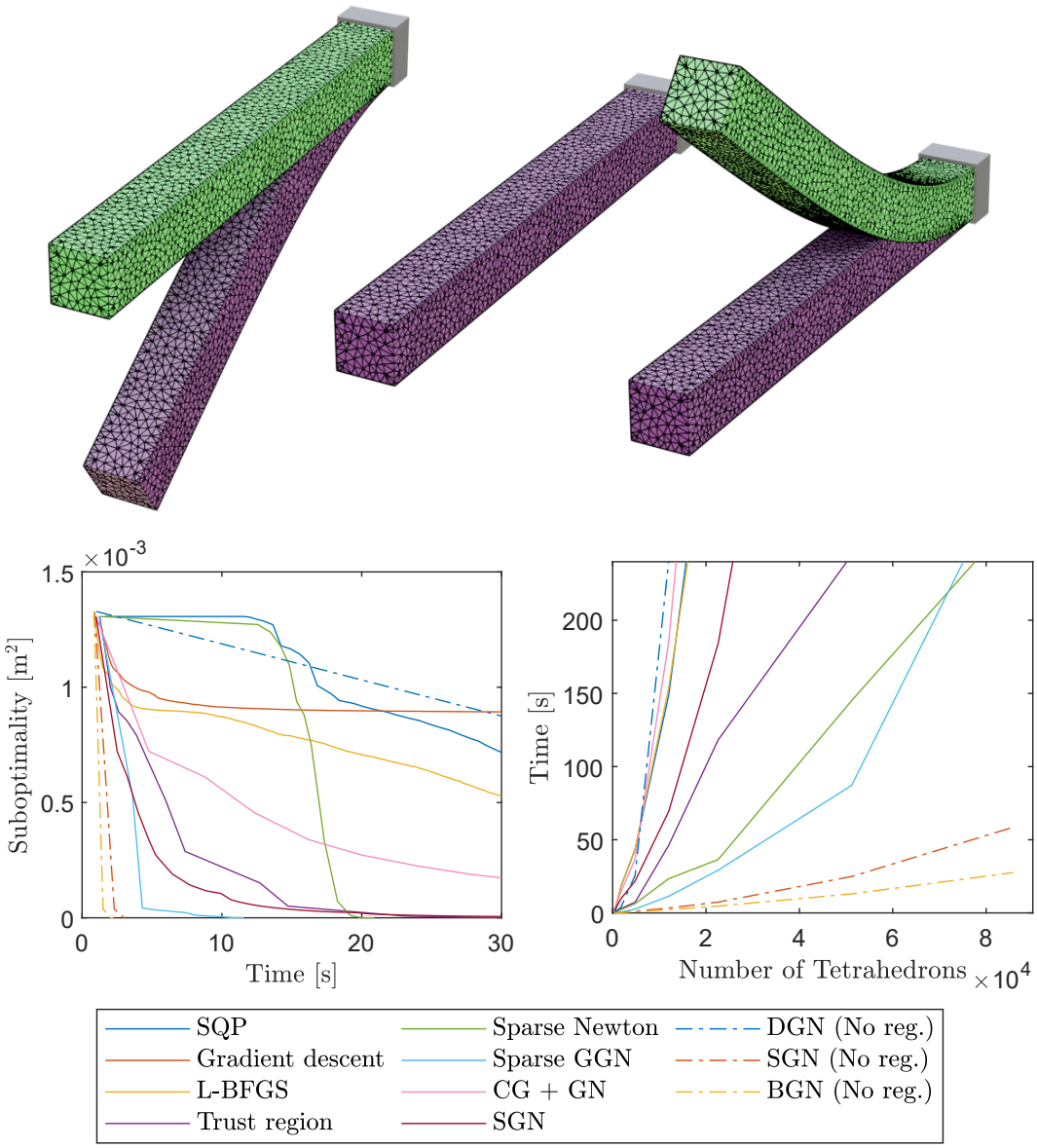


Figure 3.2 – Performance comparison for different solvers on an inverse elastic design problem. *Top left*: initial rest shape (*green*) and corresponding deformed state (*purple*). *Top middle*: target shape. *Top right*: optimized rest shape (*green*) and corresponding deformed state (*purple*). *Bottom left*: objective value vs. computation time for a mesh size of 3228 vertices. *Bottom right*: computation time vs. problem size.

the resulting equilibrium state is as close as possible to a given target shape,

$$f(\mathbf{x},\mathbf{p}) = \frac{1}{2n_x} \left\|\mathbf{x} - \mathbf{x}_{\text{target}}\right\|^2 + R(\mathbf{p}) \ . \tag{3.5.1}$$

To prevent inversions in the rest shape $\mathbf{p}$, we add a nonlinear least-squares regularizer $R(\mathbf{p})$ that penalizes per-element volume changes. For the forward simulation, we use standard linear tetrahedron elements and a Neo-Hookean material with Young's modulus $E = 10^6 Pa$

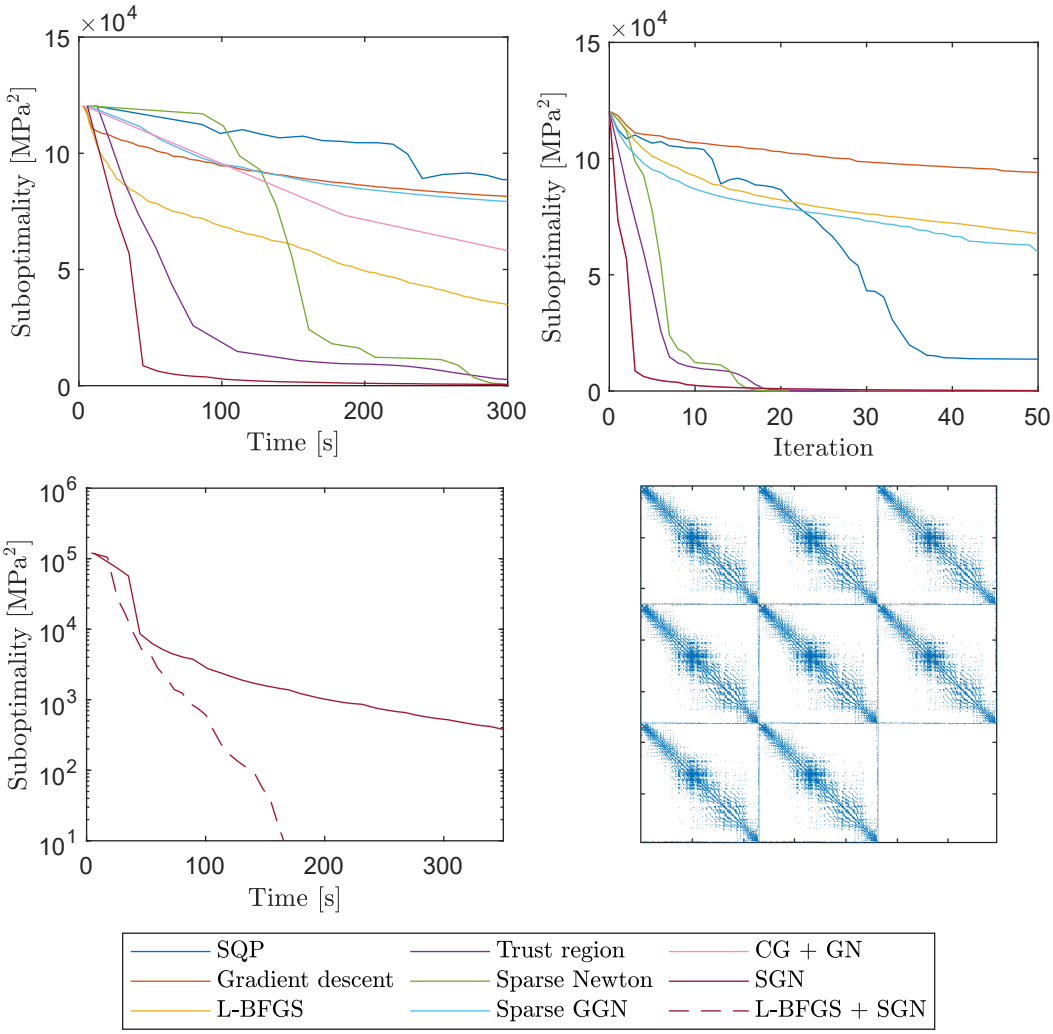


Figure 3.3 – Convergence of different solvers for the shell roof example with 15,443 vertices. Dense Gauss-Newton is not listed since the linear solver ran out of memory when computing the reduced Hessian. The sparsity pattern of the saddle point system, shown on the bottom-right, reveals a repetitive structure resulting from similar stencils for objective and constraints. The dense hessian contains $2.13 \cdot 10^9$ entries whereas the sparse KKT matrix contains $1.77 \cdot 10^7$ nonzero entries. PARDISO reported $2.18 \cdot 10^8$ nonzero entries in the decomposition.

and Poisson's ratio $\nu = 0.45$. As termination criterion we used $\|df/d\mathbf{p}\| \leq 10^{-5}$ where the gradient norm at the beginning of the optimization was close to $2 \cdot 10^{-2}$ for all resolutions.

While there are specialized solvers for gravity compensation problems [Mukherjee et al., 2018; Chen et al., 2014; Ly et al., 2018], we use this example as a benchmark for evaluating the relative performance of SGN, DGN, as well as sparse versions of full Newton and Generalized Gauss-Newton (GGN). For comparison, we also add alternative approaches based on sensitivity analysis that have recently been introduced or used in the visual computing community: the trust region solver by Panetta et al. [2019], as well as standard Gradient Descent and L-BFGS. We also include performance data for our implementation of Sequential Quadratic

Programming (SQP) using Newton's method on the KKT-conditions (see Appendix 3.A.1) and an exact L_1 merit function. As a further reference point, we also compare to the conjugate gradient method (CG) applied directly to Equation 3.4.1 and use back-substitutions to avoid forming the dense matrix. We refer to this alternative as *CG + GN*. As termination criterion for CG, we use a relative residual threshold η of 10^{-3} for all examples. We furthermore note that, when dropping the regularizer, the shape objective does not directly involve the design parameters, allowing us to apply the block Gauss-Newton (BGN) method described in Sec. 3.4.1. Interestingly, we found that the Gauss-Newton methods produce smooth rest state deformations even without regularizer, whereas all other methods led to inversions in that case. In the case of CG + GN, the default value of the threshold η was not low enough to avoid inversions and thus this method was not included.

From Fig. 3.2 (*left*) it can be seen that sparse methods without regularizer, i.e., our SGN solver and its block-solve version (BGN, outperform all other solvers by a relatively large margin. Sparse GGN and the trust region solver rank first among the alternative approaches, keeping track with SGN when using regularization. Gradient descent and L-BFGS initially perform well but progress quickly slows down. The remaining methods are not competitive on this example. The superior scaling behavior of BGN and SGN without regularizer becomes evident from Fig. 3.2 (*right*). Although the difference between BGN and SGN is small compared to the other methods, the block-solve version still offers about 30% performance increase for lower resolutions and more than 50% for higher resolutions .

3.5.2. Shell Form Finding

For the solid bar example, the objective was a simple L_2-distance on the equilibrium state which, even with regularization, did not directly couple parameters and state. In our second example, we investigate a case in which these quantities are strongly coupled—a form finding problem for a $50m \times 50m$ concrete shell roof, inspired by the works of architect Félix Candela; see [Tomas and Martí-Montrull, 2010] and Fig. 3.1. We model the roof using discrete shells [Grinspun et al., 2003] with material parameters corresponding to $E = 28GPa$, $\nu = 0.2$ as well as a density of $2500kg/m^3$. The design task consists in finding a rest shape for the shell such that the equilibrium state under gravity minimizes a stress objective in nonlinear least squares form. We use a stress model following [Gingold et al., 2004]. To encourage smooth solutions, we additionally use a regularizer $R(\mathbf{p})$ that penalizes curvature in the rest state and per-triangle deformations. The resulting objective is

$$f(\mathbf{x}, \mathbf{p}) = \sum_i ||\sigma_i(\mathbf{x}, \mathbf{p})||^2 + R(\mathbf{p}) \ , \tag{3.5.2}$$

where σ_i denotes the Cauchy stress for element i. We note that this objective couples $\mathbf{x}$ and $\mathbf{p}$, meaning that we cannot apply BGN, and its Hessian is not guaranteed to be positive-definite.

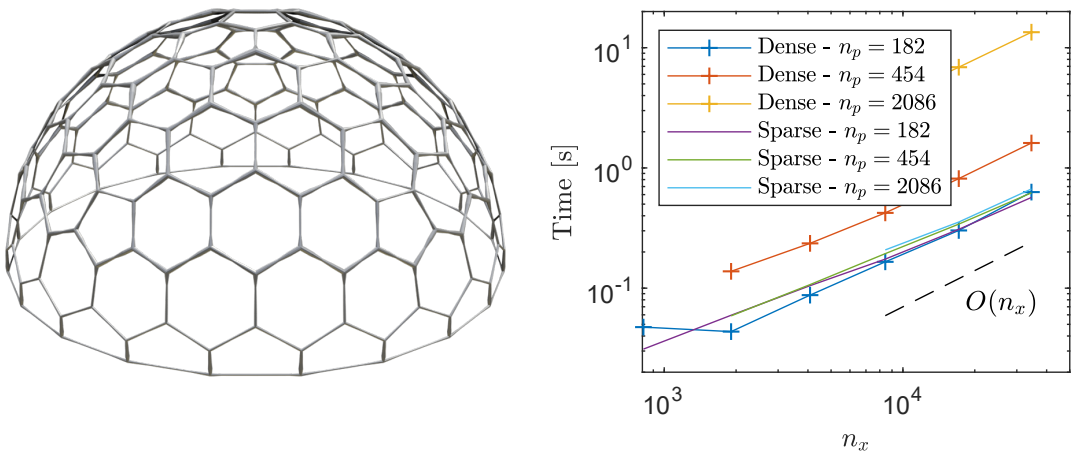


Figure 3.4 – Time to compute the search direction for the rod dome as a function of state size n_x. We subdivide the edges to add more state variables and compare timings for different numbers of parameters.

As can be seen from Fig. 3.3 (*left*), SGN clearly outperforms all other methods. Interestingly, the sparse Generalized Gauss-Newton performs similar to Gradient Descent and far worse than L-BFGS, which can be attributed to the Hessian approximation becoming indefinite.

Although SGN rapidly decreases the initial objective by almost two orders of magnitude, the log-scale plot in Fig. 3.3 (*right*) reveals that convergence slows down afterwards. However, the combination of SGN with L-BFGS as described in Sec. 3.4 is able to sustain rapid convergence in this case.

For this example, dense Gauss-Newton ran out of memory when trying to compute the dense reduced Hessian. Fig. 3.1 nevertheless compares timings between SGN and DGN for smaller problem sizes, indicating that, due to its different asymptotic complexity, SGN breaks even already for problem sizes beyond a few hundred parameters.

We note that, in order to allow for larger geometry changes, we used a lower regularization weight for the result shown in Fig. 3.1 than for the comparison shown in Fig. 3.3 since otherwise, not all methods would converge to the same solution.

3.5.3. Rod Dome

In the third example, we consider an inverse design problem for a hemispherical dome made from interconnected elastic rods. The dome is subject to a force applied at the top and we impose Dirichlet boundary conditions on the bottom. The design parameters are radii for the rods that are prescribed at connection points and interpolated along the rods. The goal, then, is to find parameters that minimize a weighted combination of the total mass of the structure—approximated as the L_2 norm of the parameter vector—and its displacement

under load. We define the design objective in nonlinear least-squares form as

$$f(\mathbf{x}, \mathbf{p}) = \frac{1}{2} \|\mathbf{x} - \mathbf{x}_{\text{undef}}\|^2 + \frac{1}{2} \|\mathbf{p}\|_V^2 + f_{\text{bounds}}(\mathbf{p}) \ , \tag{3.5.3}$$

where $f_{\text{bounds}}(\mathbf{p})$ is a log-barrier term enforcing lower and upper limits on the radii. The mapping $\|\cdot\|_V^2$ approximates the volume of the structure using conical frustums. As simulation model, we used discrete elastic rods [Bergou et al., 2010] together with the extension to rod networks by Zehnder et al. [2016] and set material parameters to $E = 69GPa$ and $\nu = 0.33$ such as to emulate aluminum rods.

The problem setup as described above allows us to independently vary the number of parameters and state variables. As can be seen from Fig. 3.4, for small numbers of parameters, DGN outperforms SGN. However, for a given number of state variables, SGN shows only a slight growth in computation time when the number of parameters is increased. The situation is very different for DGN and the break even point for this example is at around 200 parameters. It is worth noting that both dense and sparse Gauss-Newton show a similar increase in computation time with increasing number of state variables. For SGN, we conjecture that the linear scaling is due to the particular sparsity structure induced by the rod dome, which—except for the connecting nodes—exhibits a band-diagonal structure.

3.5.4. Car Control

The examples studied so far investigated the performance and scalability of our method for static equilibrium problems. We now turn to inverse dynamics problems in which we seek to optimize for control parameters such that the resulting dynamic equilibrium motion optimizes given design goals. For the first of two examples, we consider the problem of steering a simple self-driving car such as to move from a given starting position to a prescribed goal configuration. The state of the car $\mathbf{x} = (p_x, p_y, \theta)$ is described by three variables representing its position (p_x, p_y) on the plane and angle θ with respect to the first coordinate axis. The parameters $\mathbf{p}$ are control variables that include the speed v in forward direction and the steering angle s relative to the forward direction. The motion of the car is described by the simple first-order ODE $\dot{\mathbf{x}} = (v \cos\theta, v \sin\theta, v \tan s)$, which we formulate in constraint form as

$$\mathbf{c}^{t_i}(\mathbf{x}, \mathbf{p}) = \mathbf{x}^{t_i} - I_{EE}(\mathbf{x}^{t_{i-1}}, \mathbf{p}^{t_i}) \ , \tag{3.5.4}$$

where the time integration routine I_{EE} takes state variables $\mathbf{x}^{t_{i-1}}$ and control variables $\mathbf{p}^{t_i}$ at the beginning of a given time step and returns the corresponding new state $\mathbf{x}^{t_i}$. We use explicit Euler integration with a step size of $\frac{1}{30}s$ and run the simulation for N steps. The total number of state and problem variables is therefore $3N$ and $2N$, respectively. The objective that we minimize measures the difference between final and target states as

$$f(\mathbf{x}, \mathbf{p}) = \frac{w_{\text{pos}}}{2} ||\mathbf{x}^N - \mathbf{x}_{\text{target}}||^2 + \frac{w_{\text{dir}}}{2} ||\mathbf{d}(\mathbf{x}^N) - \mathbf{d}_{\text{target}}||^2 + R(\mathbf{p}) \ , \tag{3.5.5}$$

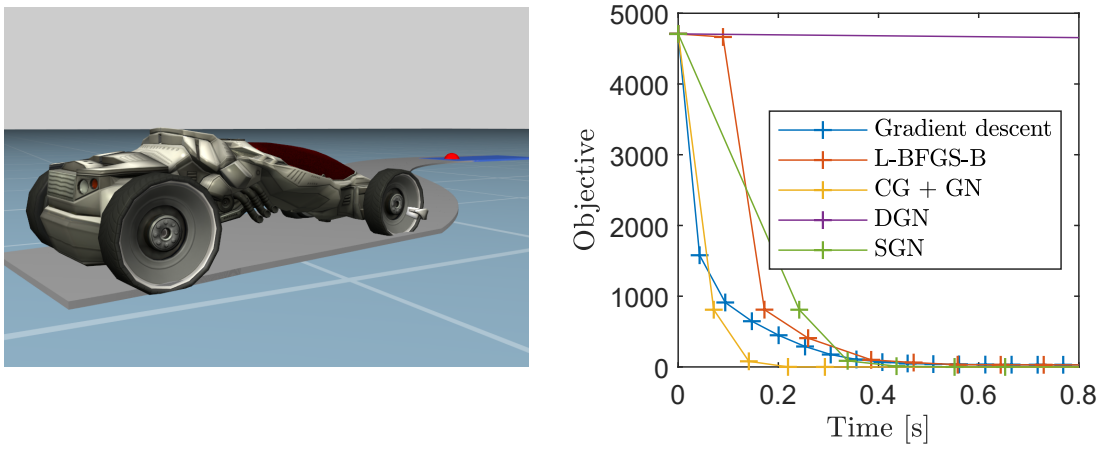


Figure 3.5 – Performance comparison for different solvers on the car example using 5000 time steps.

where $\mathbf{d}$ is the vector that points in the forward direction of the car and $R(\mathbf{p})$ is a smoothness term that penalizes differences in control variables over time. Except for L-BFGS-B, we additionally enforce bounds on the maximum velocity and steering angle by filtering the search direction.

For this relatively simple and non-stiff problem, Gradient Descent performs comparatively well and is only slightly slower than Sparse Gauss-Newton. While CG + GN outperforms all other methods in this example, the difference between sparse and dense Gauss-Newton is again substantial.

3.5.5. Cloth Control

In our second inverse dynamics example we use the method by Geilinger et al. [2020] to find time-varying handle positions for two corners of a sheet of cloth such that it moves from a given start configuration to a target state with prescribed positions. As best seen in the accompanying video, the optimized handle motion leads to two flip-overs, one in place and one with horizontal movement. As simulation model, we use a standard mass-spring system together with implicit Euler for time integration. To define the map between parameters and state for sensitivity analysis, we express the corresponding update rule in constraint form as

$$\mathbf{c}^{t_i} = \mathbf{x}^{t_i} - I_{IE}(\mathbf{x}^{t_{i-1}}, \mathbf{p}^{t_{i-1}}, \mathbf{p}^{t_i}) \tag{3.5.6}$$

where, given vertex positions $\mathbf{x}^{t_{i-1}}$ as well as control forces $\mathbf{p}^{t_{i-1}}$ and $\mathbf{p}^{t_i}$, the implicit Euler rule I_{IE} returns the new state $\mathbf{x}^{t_i}$. The cloth comprises 100 vertices and we perform N simulation steps, leading to a total of $300N$ state and $6N$ control variables. We simulate for $1.66s$ of virtual time and set the step size accordingly. The goal of matching the target state

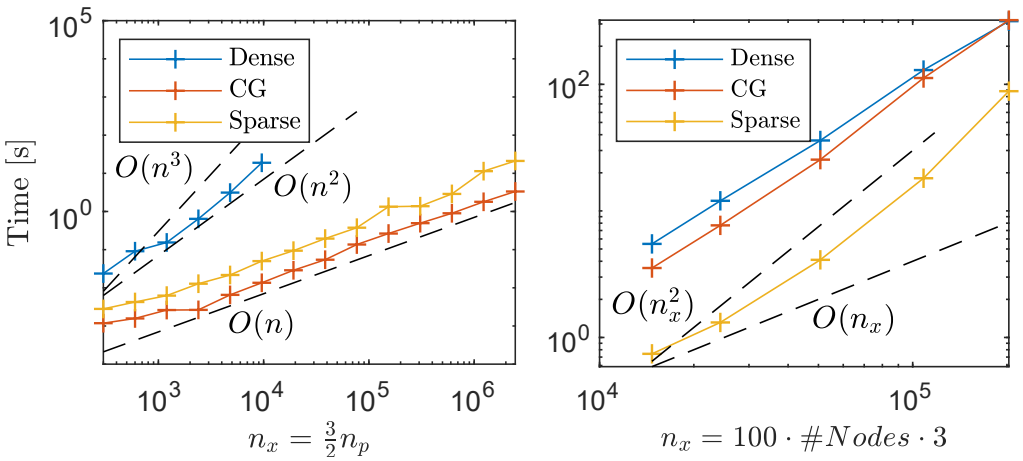


Figure 3.6 – Average time for computing Gauss-Newton search directions for the car (*left*) and cloth (*right*) control examples using the dense vs. our sparse Hessian and the CG method . *Left*: we increase the number of time steps, thus increasing problem size in terms of both state variable and parameters. *Right*: we increase the number of vertices for the cloth while keeping the number of time steps fixed and thus the number of control variables constant.

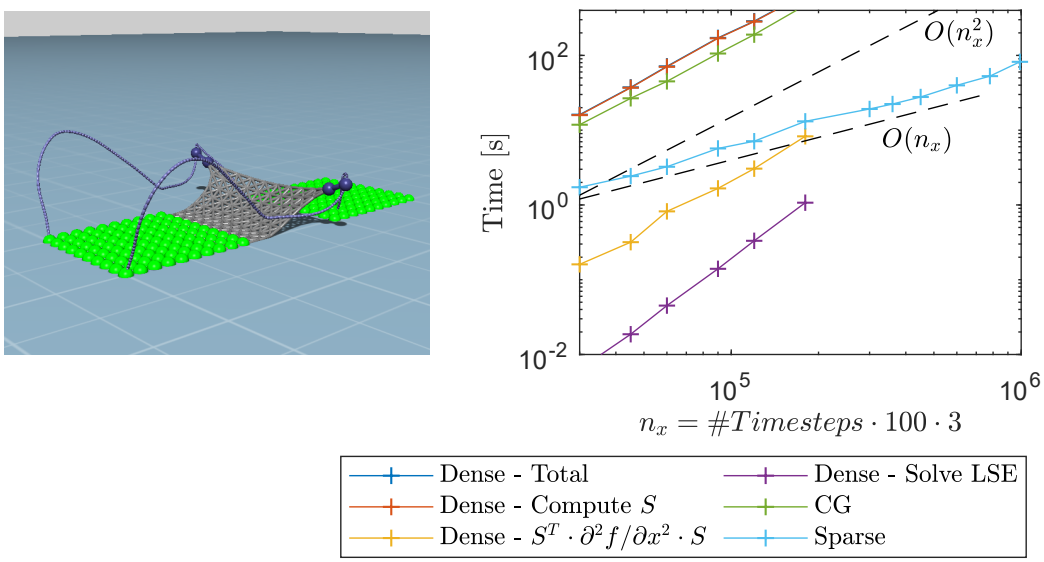


Figure 3.7 – Time required to compute the search direction for the cloth control problem. *Right*: we increase the number of time steps while keeping mesh resolution constant such that the problem size increases in terms of both state variables and parameters. The dense solution strategy ran out of memory for $n_x > 1.8 \cdot 10^5$.

is expressed as

$$f(\mathbf{x}, \mathbf{p}) = \frac{1}{2} \sum_{j \in \mathcal{S}} \left\| \mathbf{x}^j - \tilde{\mathbf{x}}^j \right\|^2 + R(\mathbf{p}) \ , \tag{3.5.7}$$

where $\mathcal{S}$ is a set of keyframes and $R(\mathbf{p})$ is a regularizer that penalizes deviations from initial handle positions, handle velocities, and cloth velocities.

For this example, we additionally provide break downs for the cost of DGN. Somewhat surprisingly, the factorization of the dense system only accounts for a fraction of the total time, which is dominated by the computation of the sensitivity matrix. Fig. 3.7 shows that SGN outperforms DGN for all problem instances, whose size we control by the number of time steps used for forward simulation.

As shown in Fig. 3.6 (*right*), when changing only the state size but keeping the number of parameters fixed, DGN scales better than SGN but breaks even only for very large problem sizes that are intractable for dense solvers on current desktop machines. The difference in scaling between SGN and DGN can be explained by the fact that, unlike for the rod dome, the cost of solving the sparse system scales quadratically with state size which, in turn, can be attributed to the higher connectivity among state variables.

3.6. Conclusions

We presented a sparse Gauss-Newton solver for sensitivity analysis that eliminates the poor performance and scaling of the dense formulation. We have shown on a diverse set of examples that SGN scales asymptotically better than its dense counterpart in almost all cases. We have furthermore provided numerical evidence that SGN outperforms existing solvers for equilibrium-constrained optimization problems on many examples.

3.6.1. Limitations & Future Work

All of our performance tests use a sparse direct solver, which imposes certain limits on the maximum problem size. One potential option for extending SGN to very large problems would be to use iterative saddle-point solvers such as the Uzawa algorithm.

Sparse Gauss-Newton transforms a dense $n_p \times n_p$ system into a sparse system of dimension $(2n_x + n_p) \times (2n_x + n_p)$, where n_x and n_p denote the number of state and design variables, respectively. This transformation is only advantageous if the number of parameters is sufficiently large. For example, when optimizing for the Young's modulus of a homogeneous elastic solid, the dense 1×1 Hessian will always be faster to invert than its sparse counterpart. At the other extreme, SGN will generally be much faster when optimizing for per-element material coefficients. While the exact break-even point depends on the problem, our experiments show that already for small to moderate n_p, SGN outperforms DGN.

Problems with sequential dependence between state variables (resulting, e.g., from time discretization) lead to a special block structure that can be leveraged to accelerate computation of the dense sensitivity matrix. We did not consider such problem-specific optimizations here.

Using a CG-based solver can be an attractive alternative, especially when fast back-substitutions are available. A disadvantage is that, for optimal performance, residual thresholds must be tweaked for each example. Furthermore, the convergence rate of CG depends

strongly on the problem. Developing specialized pre-conditioners for Eq. 3.4.1 might be an interesting option for future work.

Our formulation assumes that the objective function can be expressed in nonlinear least squares form. While not all problems exhibit this particular form, they can often be reformulated or reasonably well approximated in this way.

Some of our examples include bound constraints on the parameters, which we enforced through log-barrier penalties or by simple projection of the search direction. The latter approach, however, is neither efficient nor guaranteed to converge in the general case. Incorporating bound and inequality constraints in our formulation is an interesting direction for future work.

We did not directly analyze the impact of the cost per simulation on optimization performance. In general, problems for which forward simulation is fast will benefit more from solvers that rely only on first-order derivative information but require more function evaluations. However, we believe that our selection of examples is representative for a large range of stiff inverse problems encountered in practice. For the case of non-stiff problems, on the other hand, inexact descent methods can be an attractive alternative [Yan et al., 2018].

Finally, it would be interesting to extend our approach to efficiently compute second-order sensitivity information in the context of design space exploration for multi-objective optimization problems [Schulz et al., 2018].

Acknowledgements

This research was supported by the Discovery Accelerator Awards program of the Natural Sciences and Engineering Research Council of Canada (NSERC) and the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (Grant No. 866480). We thank the reviewers for their valuable feedback and suggestions.

3.A. Appendix

3.A.1. Equivalence Result

We show that, for general objectives, using the dense system obtained for second-order sensitivity analysis and the sparse system (3.4.4) lead to the same search direction. This proof also shows the equivalence between sensitivity analysis and sequential quadratic programming for the special case of equality constraints that are enforced to stay satisfied at all times.

Theorem 3.A.1. *Let $A, B, C, \frac{\partial c}{\partial x}, \frac{\partial c}{\partial p}$ denote sparse matrices with the correct dimensions and let $H = \frac{d\mathbf{x}}{d\mathbf{p}}^T A \frac{d\mathbf{x}}{d\mathbf{p}} + B\frac{d\mathbf{x}}{d\mathbf{p}} + \frac{d\mathbf{x}}{d\mathbf{p}}^T B^T + C$. To compute the solution to the dense linear system $H \cdot \delta\mathbf{p} = -\frac{df}{d\mathbf{p}}^T$, we can equivalently solve the larger sparse system (3.4.4).*

PROOF. See construction in Sec. 3.4. □

We next introduce the Lagrangian for the optimization problem (3.3.1) as

$$\mathcal{L}(\mathbf{x}, \mathbf{p}, \boldsymbol{\lambda}) = f(\mathbf{x}, \mathbf{p}) + \boldsymbol{\lambda}^T \mathbf{c}(\mathbf{x}, \mathbf{p}) \tag{3.A.1}$$

whose gradient is

$$\nabla \mathcal{L}(\mathbf{x}, \mathbf{p}, \boldsymbol{\lambda}) = \begin{bmatrix} \nabla_{\mathbf{x}} f + \nabla_{\mathbf{x}} \mathbf{c}^T \cdot \boldsymbol{\lambda} \\ \nabla_{\mathbf{p}} f + \nabla_{\mathbf{p}} \mathbf{c}^T \cdot \boldsymbol{\lambda} \\ \mathbf{c} \end{bmatrix} , \tag{3.A.2}$$

where $\boldsymbol{\lambda}$ are the Lagrange multipliers. The first-order optimality (or KKT) conditions correspond to $\nabla \mathcal{L}(\mathbf{x}, \mathbf{p}, \boldsymbol{\lambda}) = \mathbf{0}$. Solving these conditions with Newton's method leads to the so-called KKT system

$$\begin{bmatrix} \nabla^2_{\mathbf{xx}} f + \nabla^2_{\mathbf{xx}} \mathbf{c} : \boldsymbol{\lambda} & \nabla^2_{\mathbf{xp}} f + \nabla^2_{\mathbf{xp}} \mathbf{c} : \boldsymbol{\lambda} & \nabla_{\mathbf{x}} \mathbf{c}^T \\ \nabla^2_{\mathbf{px}} f + \nabla^2_{\mathbf{px}} \mathbf{c} : \boldsymbol{\lambda} & \nabla^2_{\mathbf{pp}} f + \nabla^2_{\mathbf{pp}} \mathbf{c} : \boldsymbol{\lambda} & \nabla_{\mathbf{p}} \mathbf{c}^T \\ \nabla_{\mathbf{x}} \mathbf{c} & \nabla_{\mathbf{p}} \mathbf{c} & 0 \end{bmatrix} \begin{bmatrix} \delta \mathbf{x} \\ \delta \mathbf{p} \\ \delta \boldsymbol{\lambda} \end{bmatrix} = - \begin{bmatrix} \nabla_{\mathbf{x}} \mathcal{L} \\ \nabla_{\mathbf{p}} \mathcal{L} \\ \mathbf{c} \end{bmatrix} ,$$

where we used the shorthand $\nabla^2_{\mathbf{yz}} \mathbf{c} : \boldsymbol{\lambda} = \sum_i \boldsymbol{\lambda}_i \nabla^2_{\mathbf{yz}} \mathbf{c}_i$.

Theorem 3.A.2. *When using the adjoint variables as Lagrange multipliers*

$$\boldsymbol{\lambda}^{\mathbf{c}=0} = - \left[\frac{\partial \mathbf{c}}{\partial \mathbf{x}} \right]^{-T} \frac{\partial f}{\partial \mathbf{x}}^T , \tag{3.A.3}$$

the KKT system and the system for second-order sensitivity analysis, $\frac{df^2}{d\mathbf{p}^2} \delta \mathbf{p} = - \frac{df}{d\mathbf{p}}^T$*, give the same search direction* $\delta \mathbf{p}$*.*

PROOF. We show that, for $\boldsymbol{\lambda} = \boldsymbol{\lambda}^{\mathbf{c}=0}$, we have

$$\begin{aligned} \frac{df^2}{d\mathbf{p}^2} &= \frac{d\mathbf{x}^T}{d\mathbf{p}} \frac{\partial^2 f}{\partial \mathbf{x}^2} \frac{d\mathbf{x}}{d\mathbf{p}} + \frac{\partial^2 f}{\partial \mathbf{x} \partial \mathbf{p}} \frac{d\mathbf{x}}{d\mathbf{p}} + \frac{d\mathbf{x}^T}{d\mathbf{p}} \frac{\partial^2 f}{\partial \mathbf{p} \partial \mathbf{x}} + \frac{\partial^2 f}{\partial \mathbf{p}^2} + \sum_i \frac{\partial f}{\partial \mathbf{x}_i} \frac{d^2 \mathbf{x}_i}{d\mathbf{p}^2} \\ &= \frac{d\mathbf{x}^T}{d\mathbf{p}} \nabla^2_{\mathbf{xx}} \mathcal{L} \frac{d\mathbf{x}}{d\mathbf{p}} + \nabla^2_{\mathbf{xp}} \mathcal{L} \frac{d\mathbf{x}}{d\mathbf{p}} + \frac{d\mathbf{x}^T}{d\mathbf{p}} \nabla^2_{\mathbf{px}} \mathcal{L} + \nabla^2_{\mathbf{pp}} \mathcal{L} \end{aligned} \tag{3.A.4}$$

Only the term involving second-order sensitivities is non-obvious. Using basic transformations, we obtain

$$\sum_k \frac{\partial f}{\partial \mathbf{x}_k} \frac{d^2 \mathbf{x}_k}{d\mathbf{p}_j d\mathbf{p}_i} = \tag{3.A.5}$$

$$\begin{aligned} &= - \sum_k \frac{\partial f}{\partial \mathbf{x}_k} \left[\left(\frac{\partial \mathbf{c}}{\partial \mathbf{x}} \right)^{-1} \left(\frac{d\mathbf{x}_m}{d\mathbf{p}_j} \frac{\partial^2 \mathbf{c}}{\partial \mathbf{x}_m \partial \mathbf{x}_l} \frac{d\mathbf{x}_l}{d\mathbf{p}_i} + \frac{\partial^2 \mathbf{c}}{\partial \mathbf{x}_m \partial \mathbf{p}_i} \frac{d\mathbf{x}_m}{d\mathbf{p}_j} + \frac{d\mathbf{x}_l}{d\mathbf{p}_i} \frac{\partial^2 \mathbf{c}}{\partial \mathbf{p}_j \partial \mathbf{x}_l} + \frac{\partial^2 \mathbf{c}}{\partial \mathbf{p}_j \partial \mathbf{p}_i} \right) \right]_k \\ &= - \sum_k \left(\left(\frac{\partial \mathbf{c}}{\partial \mathbf{x}} \right)^{-T} \frac{\partial f}{\partial \mathbf{x}}^T \right)_k \left(\frac{d\mathbf{x}_m}{d\mathbf{p}_j} \frac{\partial^2 \mathbf{c}_k}{\partial \mathbf{x}_m \partial \mathbf{x}_l} \frac{d\mathbf{x}_l}{d\mathbf{p}_i} + \frac{\partial^2 \mathbf{c}_k}{\partial \mathbf{x}_m \partial \mathbf{p}_i} \frac{d\mathbf{x}_m}{d\mathbf{p}_j} + \frac{d\mathbf{x}_l}{d\mathbf{p}_i} \frac{\partial^2 \mathbf{c}_k}{\partial \mathbf{p}_j \partial \mathbf{x}_l} + \frac{\partial^2 \mathbf{c}_k}{\partial \mathbf{p}_j \partial \mathbf{p}_i} \right) \\ &= \sum_k \boldsymbol{\lambda}_k^{\mathbf{c}=0} \left(\frac{d\mathbf{x}_m}{d\mathbf{p}_j} \frac{\partial^2 \mathbf{c}_k}{d\mathbf{x}_m \partial \mathbf{x}_l} \frac{d\mathbf{x}_l}{d\mathbf{p}_i} + \frac{\partial^2 \mathbf{c}_k}{\partial \mathbf{x}_m \partial \mathbf{p}_i} \frac{d\mathbf{x}_m}{d\mathbf{p}_j} + \frac{d\mathbf{x}_l}{d\mathbf{p}_i} \frac{\partial^2 \mathbf{c}_k}{\partial \mathbf{p}_j \partial \mathbf{x}_l} + \frac{\partial^2 \mathbf{c}_k}{\partial \mathbf{p}_j \partial \mathbf{p}_i} \right) . \end{aligned} \tag{3.A.6}$$

Eq. (3.A.4) thus holds and, using Theorem 3.A.1, the result follows directly. □

3.A.2. Block Solve

Using $B = 0$ and $C = 0$ in (3.4.4), we have

$$\delta\boldsymbol{\lambda} = -\frac{\partial \mathbf{c}}{\partial \mathbf{p}}^{-T} \frac{df}{d\mathbf{p}}^{T} \quad , \quad \text{and} \quad \delta\mathbf{x} = -A^{-1}\frac{\partial \mathbf{c}}{\partial \mathbf{x}}^{T} \delta\boldsymbol{\lambda} \ .$$

Using these definitions in the third row of (3.4.4), we obtain

$$\begin{aligned}\delta\mathbf{p} &= -\frac{\partial \mathbf{c}}{\partial \mathbf{p}}^{-1} \frac{\partial \mathbf{c}}{\partial \mathbf{x}}\delta\mathbf{x} = \frac{\partial \mathbf{c}}{\partial \mathbf{p}}^{-1} \frac{\partial \mathbf{c}}{\partial \mathbf{x}} A^{-1}\frac{\partial \mathbf{c}}{\partial \mathbf{x}}^{T} \delta\boldsymbol{\lambda} \\ &= -\frac{\partial \mathbf{c}}{\partial \mathbf{p}}^{-1} \frac{\partial \mathbf{c}}{\partial \mathbf{x}} A^{-1}\frac{\partial \mathbf{c}}{\partial \mathbf{x}}^{T} \frac{\partial \mathbf{c}}{\partial \mathbf{p}}^{-T} \frac{df}{d\mathbf{p}}^{T} \quad ,\end{aligned}$$

and using (3.3.4) with $\frac{\partial f}{\partial \mathbf{p}} = \mathbf{0}$, we finally have

$$\begin{aligned}\delta\mathbf{p} &= \frac{\partial \mathbf{c}}{\partial \mathbf{p}}^{-1} \frac{\partial \mathbf{c}}{\partial \mathbf{x}} A^{-1}\frac{\partial \mathbf{c}}{\partial \mathbf{x}}^{T} \frac{\partial \mathbf{c}}{\partial \mathbf{p}}^{-T} \left(\frac{\partial \mathbf{c}}{\partial \mathbf{p}}^{T} \frac{\partial \mathbf{c}}{\partial \mathbf{x}}^{-T}\right) \frac{\partial f}{\partial \mathbf{x}}^{T} \\ &= \frac{\partial \mathbf{c}}{\partial \mathbf{p}}^{-1} \frac{\partial \mathbf{c}}{\partial \mathbf{x}} A^{-1}\frac{\partial f}{\partial \mathbf{x}}^{T} \quad .\end{aligned}$$

3.S. Supplementary Material

3.S.1. Accuracy of linear system solves

We investigate how accurately the linear systems were solved when applying the solution strategies described in Sec. 5 of the main paper inside SGN. We track the relative residual $\|\mathbf{r}\| \,/\, \|\mathbf{b}\|$, where $\mathbf{b}$ is the right-hand side, and the weighted residual ρ [von Petersdorff, 2020] using the $1-$matrix norm. ρ provides insight on whether the residual is stemming from the condition number or from the computational method used to solve the linear system. ρ being close to or below machine precision indicates that the system has been solved without great numerical issues, in this case the relative residual can still be large because of the condition number of the matrix. The results can be seen below.

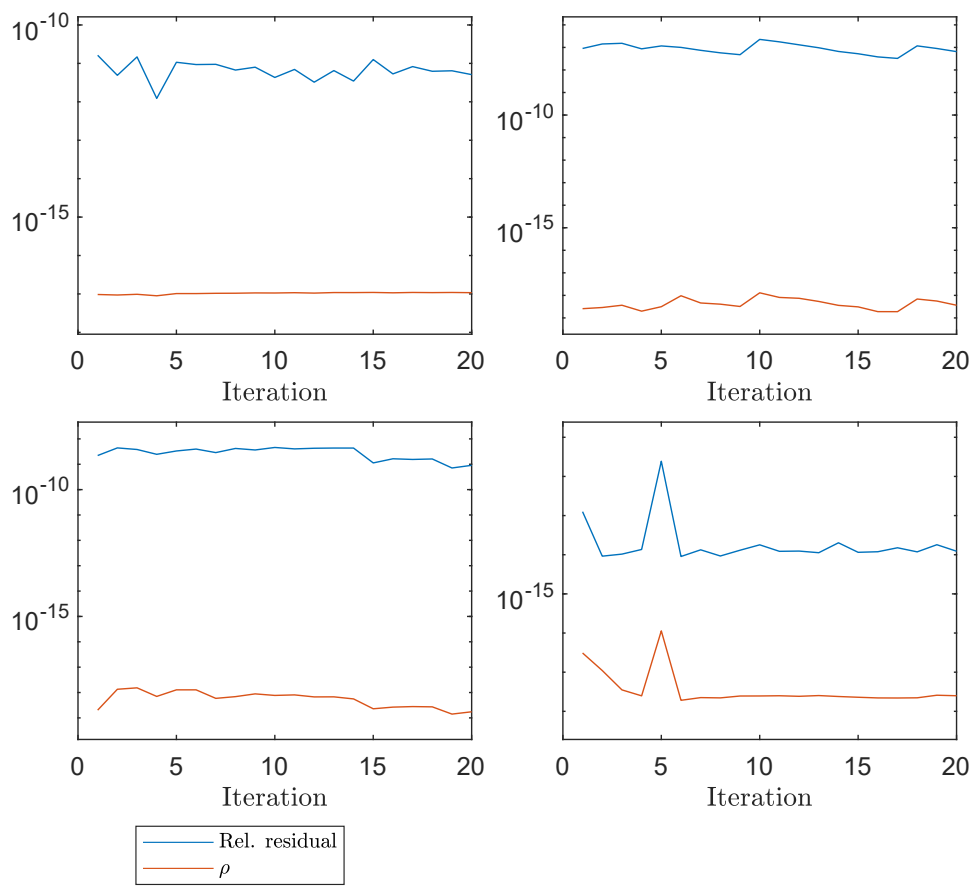


Figure 3.8 – The solution accuracy for the first 20 iterations across an optimization using SGN, from top-left to top-right, bottom-left to bottom-right: Inverse Elastic Design, Shell Form Finding, Rod Dome and Cloth Control

Chapter 4

NTopo: Mesh-free Topology Optimization using Implicit Neural Representations

by

Jonas Zehnder[1], Yue Li[2], Stelian Coros[2], and Bernhard Thomaszewski[1,2]

([1]) Université de Montréal

([2]) ETH Zürich

Publication

This article has been submitted to the Conference on Neural Information Processing Systems (NeurIPS 2021) and has been accepted to be presented at a poster session. The article was reformatted to fit the style of this thesis.

Jonas Zehnder and Yue Li are the two principal authors of this project. They both contributed to the method, implementation and collection of results. All authors collaborated on the writing of the article.

Abstract. Recent advances in implicit neural representations show great promise when it comes to generating numerical solutions to partial differential equations. Compared to conventional alternatives, such representations employ parameterized neural networks to define, in a mesh-free manner, signals that are highly-detailed, continuous, and fully differentiable. In this work, we present a novel machine learning approach for topology optimization—an important class of inverse problems with high-dimensional parameter spaces and highly nonlinear objective landscapes. To effectively leverage neural representations in the context of mesh-free topology optimization, we use multilayer perceptrons to parameterize both density and displacement fields. Our experiments indicate that our method is highly competitive for minimizing structural compliance objectives, and it enables self-supervised learning of continuous solution spaces for topology optimization problems.
Keywords: implicit representations, topology optimization, neural networks, multilayer perceptrons, mesh-free

4.1. Introduction

Deep neural networks are starting to show their potential for solving partial differential equations (PDEs) in a variety of problem domains, including turbulent flow, heat transfer, elastodynamics, and many more [Hennigh et al., 2020; Raissi et al., 2019; Rao et al., 2020; Sitzmann et al., 2020; Lu et al., 2019]. Thanks to their smooth and analytically-differentiable nature, implicit neural representations with periodic activation functions are emerging as a particularly attractive and powerful option in this context [Sitzmann et al., 2020]. In this work, we explore the potential of implicit neural representations for structural topology optimization—a challenging inverse elasticity problem with widespread application in many fields of engineering [Bendsoe and Sigmund, 2013].

Topology optimization (TO) methods seek to find designs for physical structures that are as stiff as possible (i.e. least compliant) with respect to known boundary conditions and loading forces while adhering to a given material budget. While TO with mesh-based finite element analysis is a well-studied problem [Bendsøe and Sigmund, 1995], we argue that mesh-free methods provide unique opportunities for machine learning. We propose the first self-supervised, fully mesh-free method based on implicit neural representations for topology optimization. The core of our approach is formed by two neural networks: a displacement network representing force-equilibrium configurations that solve the forward problem, and a density network that learns optimal material distributions in the domain of interests. To leverage the power of these representations, we cast TO as a stochastic optimization problem using Monte Carlo sampling. Compared to conventional mesh-based TO, this setting introduces new challenges that we must address. To account for the nonlinear nature of implicit neural representations, we introduce a convex density-space objective that guides the neural network towards desirable solutions. We furthermore introduce several

concepts from FEM-based topology optimization methods into our learning-based Monte Carlo setting to stabilize the training process and to avoid poor local minima.

We evaluate our method on a set of standard TO problems in two and three dimensions. Our results indicate that neural topology optimization with implicit representations is able to match the performance of state-of-the-art mesh-based solvers. To further explore the potential advantages of this approach over conventional methods, we show how our formulation enables self-supervised learning of continuous solution spaces for this challenging class of problems.

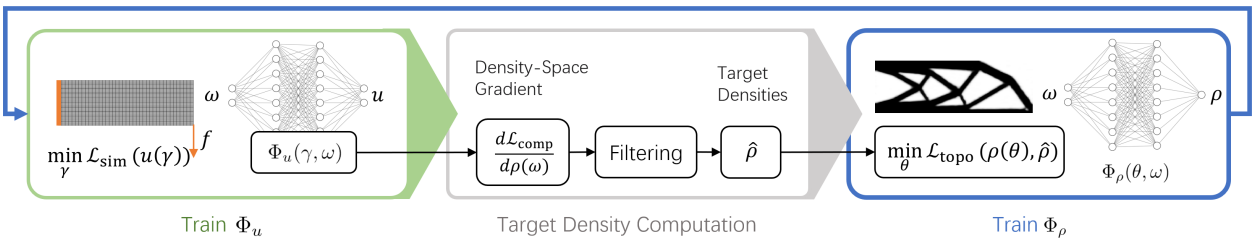


Figure 4.1 – Neural topology optimization pipeline. We compute optimal material distributions by alternately training two neural networks: the displacement network Φ_u and the density network Φ_ρ, mapping spatial coordinates ω to equilibrium displacements u and optimal densities ρ, respectively. In each iteration, we first update Φ_u by minimizing the total potential energy of the system. We then perform sensitivity analysis to compute density-space gradients which, after applying our sensitivity filtering, give rise to target density fields $\hat{\rho}$. Finally, we update Φ_ρ by minimizing the convex objective $\mathcal{L}_{\text{topo}}$ based on mean squared error between current and target densities.

4.2. Related work

Neural Networks for Solving PDEs. Deep neural networks have been widely used in different fields to provide solutions for partial differential equations for both forward simulation and inverse design problems [Hennigh et al., 2020; Sitzmann et al., 2020; Raissi et al., 2020]. In this context, PDEs can be solved either in their strong form [Dissanayake and Phan-Thien, 1994; Lagaris et al., 1998; Sirignano and Spiliopoulos, 2018] or variational form [He et al., 2018; Weinan and Yu, 2018]. We refer to DeepXDE [Lu et al., 2019] for a detailed review. Explorations into using deep learning alongside conventional solvers for simulation have been conducted with the goal of accelerating computations [Xue et al., 2020] or learning the governing physics [Holl et al., 2020; Battaglia et al., 2016; Sanchez-Gonzalez et al., 2020; Pfaff et al., 2020; Grzeszczuk et al., 1998; Holden et al., 2019]. With their continuous and analytically-differentiable solution fields, neural implicit representations with periodic activation functions [Sitzmann et al., 2020] offer a promising alternative to mesh-based finite element analysis. We leverage this new representation to solve high-dimensional inverse elasticity problems in a fully mesh-free manner.

Differentiable Simulation for Machine Learning. There is growing interest in differentiable simulation methods that enable physics-based supervision in learning frameworks [Hu et al., 2019b; Liang et al., 2019; Hu et al., 2019a; Geilinger et al., 2020; Um et al., 2020; Holl et al., 2020]. Liang *et al.* [Liang et al., 2019] proposed a differentiable cloth simulation method for optimizing material properties, external forces, and control parameters. Hu *et al.* [Hu et al., 2019b] targets reinforcement learning problems with applications in soft robotics. Geilinger *et al.* [Geilinger et al., 2020] proposed an analytically differentiable simulation framework that handles frictional contacts for both rigid and deformable bodies. To reduce numerical errors in a traditional solver Um *et al.* [Um et al., 2020] leverage a differentiable fluid simulator inside the training loop. Similar to these existing methods, our approach relies on differentiable simulation at its core, but targets mesh-free, stochastic integration for elasticity problems.

Neural Representations. Using implicit neural representation for complex signals has been an on-going topic of research in computer vision [Saito et al., 2020; Park et al., 2019], computer graphics [Mildenhall et al., 2020; Tancik et al., 2020], and engineering [Sitzmann et al., 2020; Hennigh et al., 2020]. PIFu [Saito et al., 2020] learns implicit representations of human bodies for monocular shape estimation. Sitzmann *et al.* [Sitzmann et al., 2020] use MLPs with sinusoidal activation functions to represent signed distance fields in a fully-differentiable manner. Takikawa *et al.* [Takikawa et al., 2021] introduced an efficient neural representation that enables real-time rendering of high-quality surface reconstructions. Mildenhall *et al.* [Mildenhall et al., 2020] describe a neural radiance field representation for novel view synthesis. We leverage the advantages of implicit neural representations, demonstrated in these previous works, to learn the high-dimensional solutions of topology optimization problems.

Topology Optimization with Deep Learning. Topology optimization methods aim to find an optimal material distribution of a design domain given a material budget, boundary conditions, and applied forces. Building on a large body of finite-element based methods [Bendsoe and Kikuchi, 1988; Bendsoe and Sigmund, 2013; Wang et al., 2003; Sigmund, 2007; Bendsøe and Sigmund, 1999; Bendsøe, 1989; Andreassen et al., 2011; Sigmund, 2001; Allaire et al., 2004, 2005; Guo et al., 2014; Zhang et al., 2016, 2018; Luo et al., 2008; van Dijk et al., 2013], recent efforts have explored the use of deep learning techniques in this context. One line of work leverages mesh-based simulation data and convolutional neural networks (CNNs) to accelerate the optimization process [Zhang et al., 2019; Yu et al., 2019; Banga et al., 2018; Ulu et al., 2016; Nie et al., 2020; Lei et al., 2019; Lin et al., 2018]. Perhaps closest to our work is the method by Hoyer *et al.* [Hoyer et al., 2019], who reparameterize design variables with CNNs, but use mesh-based finite element analysis for simulation. While Hoyer *et al.* [Hoyer et al., 2019] map latent vectors to discrete grid densities, Chandrasekhar *et al.* [Chandrasekhar and Suresh, 2020] use multilayer perceptrons to learn a continuous

mapping from spatial locations to density values. However, as we show in our comparisons, their choice of using ReLU activation functions leads to overly simplified solutions whose structural performance is not on par with results from conventional methods [Liu et al., 2018; Li et al., 2021; Aage et al., 2017]. In addition, both methods use an explicit mesh for forward simulation and sensitivity analysis, whereas our method is entirely based on neural implicit representations.

4.3. Problem Statement and Overview

Given applied forces, boundary conditions, and a target material volume ratio $\hat{V}$, the goal of topology optimization is to find the material distribution that leads to the stiffest structure. This task can be formulated as a constrained bilevel minimization problem,

$$\begin{aligned} &\mathcal{L}_{\text{comp}}(\rho) = \int_\Omega e(\rho, u, \omega)\, d\omega \\ &\text{s. t. } u(\rho) = \arg\min_{u'} \mathcal{L}_{\text{sim}}(u', \rho), \qquad \rho(\omega) \in \{0, 1\}, \qquad \frac{1}{|\Omega|}\int_\Omega \rho\, d\omega = \hat{V}\ , \end{aligned} \tag{4.3.1}$$

where the loss $\mathcal{L}_{\text{comp}}$ measures how compliant the material is, ρ is the material density field, ω runs over the domain Ω and $|\Omega|$ is the volume of Ω. The displacement u is a result of minimizing the simulation loss $\mathcal{L}_{\text{sim}}$, ensuring the configuration is in force equilibrium. e is the pointwise compliance, which is equivalent to the internal energy up to a constant factor and is measuring how much the material is deformed under load; see Section 4.4.1 for details. Although manufacturing typically demands binary material distributions, densities are often allowed to take on continuous values while convergence to binary solutions is encouraged. We follow the same strategy and parameterize densities ρ and displacements u using implicit neural representations, $\Phi_\rho(\omega; \theta)$ and $\Phi_u(\omega; \gamma)$, respectively. By sampling a batch of locations ω^b and $\rho^b = \rho(\omega^b)$, we compute an estimate of $\mathcal{L}_{\text{comp}}$ using Monte Carlo integration, $\mathcal{L}_{\text{comp}} \approx \frac{|\Omega|}{n}\sum_i^n e(\omega_i^b)$. If u is a displacement in force equilibrium, we can compute the total gradient of the compliance loss with respect to the densities of the batch as

$$s = \frac{d\mathcal{L}_{\text{comp}}}{d\rho} = \frac{\partial \mathcal{L}_{\text{comp}}}{\partial \rho} + \frac{\partial \mathcal{L}_{\text{comp}}}{\partial u}\frac{du}{d\rho} = -\frac{\partial \mathcal{L}_{\text{comp}}}{\partial \rho}\ . \tag{4.3.2}$$

We will refer to this expression as the *density-space gradient*; see the supplemental document for a detailed derivation. The density-space gradient indicates how the compliance loss changes w.r.t. the density values, assuming that the force equilibrium constraints remain satisfied. On this basis, we compute the total gradient of the compliance loss with respect to the neural network parameters as $\frac{d\mathcal{L}_{\text{comp}}}{d\theta} = \frac{d\mathcal{L}_{\text{comp}}}{d\rho}\frac{\partial \rho}{\partial \theta}$.

Using this gradient together with a penalty on the volume constraint would be one potential option for solving Equation (4.3.1). In practice, however, we observed that this approach does not lead to satisfying behavior. We elaborate on this problem below. TO is a

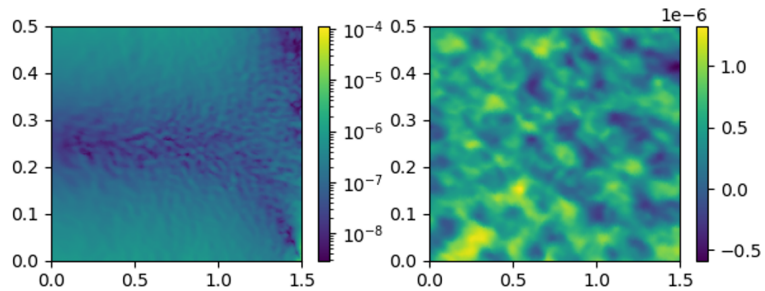


Figure 4.2 – Failure of naive gradient descent. *Left*: negative density-space gradient on the domain for the beam example in the first iteration. Log-scale coloring is used to emphasize structure. *Right*: after one step of gradient descent (learning rate 10^{-5}), the neural network output has lost all structure from the density-space gradient.

non-convex optimization problem, whose solutions depend on the optimization method [Hoyer et al., 2019] and the path density values take. Much research has been done on developing optimization strategies that converge to *good* local minima for FEM-based solvers. The density-space gradient is generally considered a good update direction for mesh-based approaches. However, when using standard optimizers to update the parameters for our density network, the resulting change in densities is not at all aligned with the density-space gradient; see Figure 4.2. We tested both ADAM [Kingma and Ba, 2014] and SGD with little difference in the results—eventually, both approaches converge to local minima that are meaningless from a structural point of view; see Section 4.5.4.

To analyze this unexpected behavior, we consider the change in densities induced by a step in the negative direction of the density-space gradient $\Delta\rho = -\alpha \frac{d\mathcal{L}_{\text{comp}}}{d\rho}$, where α is the learning rate. The corresponding first-order change in network parameters $\Delta\theta$ is $-\alpha \frac{d\mathcal{L}_{\text{comp}}}{d\rho} \frac{\partial\rho}{\partial\theta}$. However, using this parameter change in the Taylor series expansion of the network output, we obtain

$$\Delta\rho' = \rho(\theta + \Delta\theta) - \rho(\theta) = -\alpha \frac{d\mathcal{L}_{\text{comp}}}{d\rho} \frac{\partial\rho}{\partial\theta} \frac{\partial\rho}{\partial\theta}^T + O(\alpha^2) \ . \tag{4.3.3}$$

It is evident that, unless the Jacobian $\partial\rho/\partial\theta$ of the neural network is a unitary matrix, the direction of density change $\Delta\rho'$ is different from $\Delta\rho$ even as $\alpha \to 0$.

To avoid converging to bad local minima, we seek to update the network parameters such that the network output changes along the density-space gradient. To this end, we define point-wise density targets $\hat{\rho}^b$ indicating how the network output should change. We then minimize the convex loss function

$$\mathcal{L}_{\text{topo}}(\theta) = \frac{1}{n_b n} \sum_{b}^{n_b} \sum_{j}^{n} \left\| \rho(\omega_j^b; \theta) - \hat{\rho}(\omega_j^b) \right\|^2 \ . \tag{4.3.4}$$

While our density-space optimization strategy greatly improves results, we have still observed convergence to minima with undesirable artefacts. Drawing inspiration from mesh-based TO methods [Bourdin, 2001], we solve this problem with a sensitivity filtering approach

(termed $\mathcal{FT}$ in Algorithm 1). To further accelerate convergence, we also adapt the optimality criterion method [Bendsøe and Sigmund, 1995] to our setting ($\mathcal{OC}$ in Algorithm 1). Our resulting neural topology optimization algorithm, which we dubbed *NTopo*, is summarized in Algorithm 1 and further explained in the following Section.

Algorithm 1 NTopo: Neural topology optimization.

Initialize $\gamma^{(-1)}, \theta^{(0)}$ for Φ_ρ, Φ_u; Initialize two optimizers: $\text{opt}_\theta, \text{opt}_\gamma$
Run initial simulation: $\gamma^{(0)} \leftarrow \arg\min_\gamma \mathcal{L}_{\text{sim}}(\gamma, \theta^{(0)})$
for n_{opt} iterations **do**
  for n_{sim} iterations **do** $\gamma^{(l+1)} \leftarrow \text{opt}_\gamma.\text{step}(\gamma^{(l)}, \partial\mathcal{L}_{\text{sim}}/\partial\gamma)$ **end for**
  for batch $b = 1, .., n_b$ **do** compute ρ^b, s^b **end for**
  $\hat{s} \leftarrow \mathcal{FT}(\rho^{1,\dots,n_b}, s^{1,\dots,n_b})$
  $\hat{\rho} \leftarrow \mathcal{OC}(\rho^{1,\dots,n_b}, \hat{s}^{1,\dots,n_b}, \hat{V})$
  for batch $b = 1, .., n_b$ **do** $\theta^{(l+1)} \leftarrow \text{opt}_\theta.\text{step}(\theta^{(l)}, \hat{\rho}^b, \partial\mathcal{L}_{\text{topo}}/\partial\theta)$ **end for**
end for
return Φ_ρ

4.4. Neural Topology Optimization

We start the technical description of our algorithm with a brief overview of the neural representation that we use. We then explain how to compute equilibrium configurations and how to update the density field. Finally, we introduce an extension of our method from individual solutions to entire solution spaces for a given continuous parameter range.

We use SIREN [Sitzmann et al., 2020] as neural representation, which is a dense multilayer perceptron (MLP) with sinusoidal activation functions. A SIREN-MLP with l layers, $l-1$ hidden layers, and h neurons in each hidden layer is defined as

$$\Phi(x) = W_l(\phi_{l-1} \circ \phi_{l-2} \circ \ldots \circ \phi_0)(\omega_0 x) + b_l, \qquad \phi_i(x) = \sin(W_i x_i + b_i) \tag{4.4.1}$$

where x is the input vector, y is the output vector, W_i are weight matrices, b_i are biases, and ω_0 is a frequency dependent factor. We use a standard five-layer MLP with residual links and no regularization. The weights of all layers are initialized as suggested by Sitzmann *et al.* [Sitzmann et al., 2020].

4.4.1. Computing Static Equilibrium Solutions

We parametrize the displacement field u using a neural network $\Phi_u(\omega; \gamma)$ with network weights γ. To find the displacement field u in static equilibrium, we minimize the variational form of linear elasticity which is given by

$$\min_\gamma \mathcal{L}_{\text{sim}}(u(\gamma)) = \min_\gamma \int_\Omega \frac{1}{2}\varepsilon(u) : \sigma(u) - u^T f \, d\omega \qquad \text{s.t.} \quad u|_{\partial\Omega_D} = u_D \tag{4.4.2}$$

where $\partial\Omega_D$ is the part of the boundary of the domain with prescribed displacement u_D. The internal energy in linear elasticity is given through $\frac{1}{2}\,\varepsilon : \sigma$, where the tensor contraction ":" is defined as $\epsilon : \sigma = \sum_{ij} \epsilon_{ij}\sigma_{ij}$. In two dimensions we compute the stress tensor under plane stress assumption $\sigma = (E/\,(1-\nu^2))\,((1-\nu)\varepsilon + \nu\,\mathrm{trace}(\varepsilon)\mathbf{I}\,)$ where ν is Poisson's ratio, E is the Young's modulus, $\mathbf{I} \in \mathbb{R}^{2\times 2}$ is the identity matrix and $\varepsilon = \left(\nabla u + \nabla u^T\right)/2$ is the linear Cauchy strain. In 3D we use Hooke's law $\sigma = \lambda\,\mathrm{trace}(\varepsilon)\mathbf{I} + 2\mu\epsilon$ where $\lambda = E\nu/((1+\nu)(1-2\nu))$ and $\mu = E/(2(1+\nu))$. Following SIMP [Sigmund, 2001], we parameterize the Young's modulus E using the density field as $E(\rho) = \rho^p E_1$. Larger values for the exponent p together with the volume constraint encourage binary solutions.

Sampling. To evaluate the integral in Equation (4.4.2) we resort to a Quasi-Monte Carlo sampling approach. We generate stratified samples on a grid with $n_x \times n_y$ cells in 2D (and $n_x \times n_y \times n_z$ in 3D). We adjust n_x, n_y, n_z to match the aspect ratio of the domain.

Enforcing Dirichlet Boundary Conditions. By constructing a function d that is zero on the fixed boundary and an interpolator of the function u_D, we enforce the displacement field $u(\omega) = d(\omega)\Phi_u(\omega;\gamma) + (I \circ u_D)(\omega)$ to always satisfy the essential boundary conditions, thus turning the constrained problem into an unconstrained one [McFall and Mahan, 2009]. We use simple boundaries in our examples for which analytic functions d are readily available; see the supplemental document for more details.

4.4.2. Density Field Optimization

We reparameterize the density field using a neural network Φ_ρ which maps spatial locations to their corresponding density values. The bound constraints on the densities are enforced by applying a scaled logistic sigmoid function to the network output, specifically $\rho(\omega) = \mathrm{sigmoid}(5\,\Phi_\rho(\omega))$. The total volume constraint is satisfied by the optimality criterion method described below.

Moving Mean Squared Error (MMSE).. Equation (4.3.4) can be interpreted as a mean squared error

$$\frac{1}{|\Omega|}\int_\Omega ||\rho(\omega;\theta) - \hat{\rho}(\omega)||^2\, d\omega\ . \tag{4.4.3}$$

We minimize this loss using a mini-batch gradient descent strategy, where we use every batch only once. We collect multiple batches of data from $\hat{\rho}$ before we update ρ and $\hat{\rho}$, specifically $\hat{\rho}$ changes once every outer iteration. For this reason, we refer to this updating scheme as moving mean squared error in the following.

Sensitivity filtering $\mathcal{FT}$. In conventional TO algorithms, filtering is an essential component for discovering desirable minima that avoid artefacts such as checkerboarding [Sigmund, 2007]. While our neural representation does not suffer from the same discretization limitations that give rise to checkerboard patterns, we have nevertheless observed convergence to undesirable

minima. Indeed, the neural representation alone does not remove the inherent reason for such artefacts: TO is an underconstrained optimization problem with a high-dimensional null-space. Isolating good solutions from this null-space requires additional priors, filters, or other regularizing information.

In order to address this problem, we propose a *continuous* sensitivity filtering approach that, instead of using discrete approximations [Bourdin, 2001], is based on continuous convolutions. Following this strategy, we obtain filtered sensitivities as

$$\hat{s}(\omega) = \frac{\int H(\Delta\omega)\rho(\omega+\Delta\omega)s(\omega+\Delta\omega)\,d\Delta\omega}{\max(\epsilon,\rho(\omega))\int H(\Delta\omega)\,d\Delta\omega} \tag{4.4.4}$$

where ϵ is set to 10^{-3} and H is the kernel $H(\Delta\omega) = \max(0, r - \|\Delta\omega\|)$, with radius r. Since the samples ω^i are distributed inside a grid, we can compute an approximation to the continuous filter as

$$\hat{s}^i_j = \frac{\sum_{k\in N^j} H(\omega^i_j - \omega^i_k)\rho^i_k s^i_k}{\max(\epsilon,\rho^i_j)\sum_{k\in N^j} H(\omega^i_j - \omega^i_k)} \tag{4.4.5}$$

where the neighborhood N is defined by cell sizes and radius r such that points inside the footprint of the kernel H are in the neighborhood N. Although this approximation is not an unbiased estimator of Equation (4.4.4), it led to satisfying results in all our experiments.

Multi Batch-based Optimality Criterion Method $\mathcal{OC}$. We leverage the optimality criterion method [Andreassen et al., 2011] to compute density targets that automatically satisfy volume constraints, thus avoiding penalty functions or other constraint enforcement mechanisms. To this end, we extend the discrete, mesh-based formulation to the continuous Monte-Carlo setting. One chooses a Lagrange multiplier λ such that the volume constraint is satisfied after the variables have been updated. Since it is computationally infeasible to compute the constraint exactly, we choose to satisfy the constraint in terms of its estimator using the collected batches. Additionally, we adopt a heuristic updating scheme very similar to the ones proposed by other authors [Andreassen et al., 2011], which leads to the following scheme: First a set of multiplicative updates B^i are computed, which then are applied to compute the target densities $\hat{\rho}^i = \text{clamp}(\rho^i(B^i)^\eta, \max(0,\rho^i - m), \min(1,\rho^i + m))$, where m limits the maximum movement of a specific target density and η is a damping parameter. We used $m = 0.2$ and $\eta = 0.5$. The updated B^i are computed using $B^i = -\hat{s}^i/(\lambda\frac{\partial V}{\partial \rho^i})$ where λ is found using a binary search such that the estimated volume of the updated densities using Monte Carlo integration matches the desired volume ratio across all batches $\frac{1}{n_b n}\sum_b^{n_b}\sum_j^n \hat{\rho}^b_j = \hat{V}$.

4.4.3. Continuous Solution Space

Apart from solving individual TO problems for fixed boundary conditions and material budgets, our method can be readily extended to learn entire spaces of optimal solutions, *e.g.*, a continuous range of material volume constraints $\{\hat{V}^i\}$ or boundary conditions such as force

locations. To this end, we seek to find a density function $\rho(\omega, q)$ which yields the optimal density at any point ω in the domain for any parameter vector q representing, e.g., material volume ratio $q := \hat{V}$ in the target range $Q = [\underline{V}, \overline{V}]$. In a supervised setting, a common approach is to first compute k solutions corresponding to different parameters $\{q^k\}$ and then fit the neural network using a mean squared error. By contrast, our formulation invites a fully self-supervised approach based on a modified moving mean squared error,

$$\frac{1}{|Q||\Omega|} \int_Q \int_\Omega \|\rho(\omega, q; \theta) - \hat{\rho}(\omega, q)\|^2 \, d\omega dq \ . \tag{4.4.6}$$

We minimize this loss by sampling q at random and then update the density network using the same method as described for the single target volume case.

4.5. Results

To analyze our method and evaluate its results, we start by comparing material distributions obtained with our approach on a set of 2D examples. We demonstrate the effectiveness of our method through comparisons to a state-of-the-art FEM solver (Section 4.5.1). We then investigate the ability of our formulation to learn continuous spaces of solutions in a fully self-supervised manner. Comparisons to a data-driven, supervised approach indicate better efficiency for our method, suggesting that our approach opens new opportunities for design exploration in engineering applications (Section 4.5.2). We then turn to TO problems with non-trivial boundary conditions and demonstrate generalization to 3D examples (Section 4.5.3). An ablation study justifies our choices of using sensitivity filtering and casting the nonlinear topology objective into an MMSE form (Section 4.5.4). We further compare the impact of different activation functions and provide detailed descriptions of our learning settings (Section 4.5.5).

4.5.1. Comparisons with FEM Solutions

We demonstrate that our results are competitive to those produced by a SIMP, a reference FEM approach [Andreassen et al., 2011] for mesh-based topology optimization. As can be seen in Figure 4.3, results are qualitatively similar, but our method often finds more complex supporting structures that lead to lower compliance values (see Table 4.1). To put these results in perspective, there is no topology optimization method fundamentally better than SIMP, despite decades of research. For fair comparisons, the compliance values of these structures are evaluated using the FEM solver and we consistently use fewer degrees of freedom (DoFs). The DoFs in these two methods are the number of network weights and the number of finite element cells, respectively. As can be seen, our method is more computationally expensive, but we would like to emphasize that the goal of our approach is not to outperform conventional solvers for single-solution cases, but rather find insight

for a learning-based and fully mesh-free approach to topology optimization that allows for self-supervised learning of solution spaces. We refer to the supplemental material for further details of these examples.

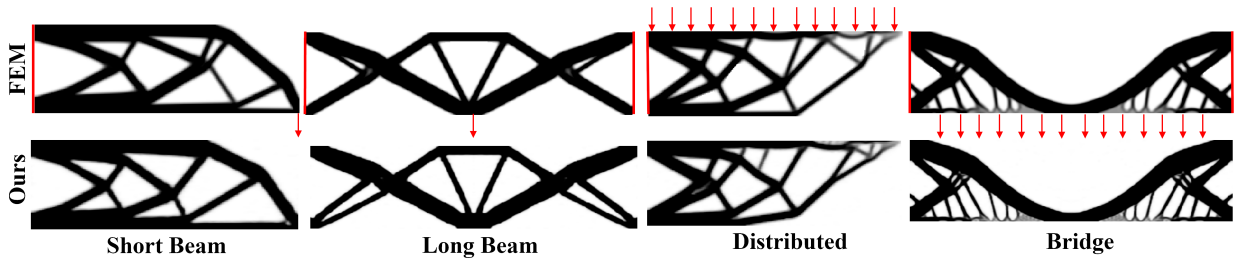


Figure 4.3 – 2D comparison on four example problems. Solutions produced by our method are qualitatively equivalent to corresponding FEM solutions. Force and Dirichlet boundary conditions are visualized in the top-row images.

Table 4.1 – Statistics of 2D comparisons. Our results achieve quantitatively lower compliance values (Comp.) than the reference solver (SIMP) for all examples.

	FEM				Ours			
Exp.	Comp.	DoFs	Iter.	Time	Comp.	DoFs	Iter.	Time
ShortBeam	1.173×10^{-3}	30,000	122	34 s	1.166×10^{-3}	28,300	200	33 min
LongBeam	2.595×10^{-4}	31,250	155	47 s	2.592×10^{-4}	28,300	200	33 min
Distributed	2.026×10^{1}	30,000	454	114 s	2.012×10^{1}	28,300	400	66 min
Bridge	4.441×10^{0}	31,250	233	72 s	4.385×10^{0}	28,300	200	33 min

4.5.2. Learning Continuous Solution Spaces

Optimal designs for different volume constraints. Here we demonstrate the capability of our method to learn continuous spaces of optimal density distributions for a continuous range of material volume constraints. We minimize the objective defined in Equation (4.4.6) for volume constraint samples drawn randomly from the range $[30\%, 70\%]$. To evaluate the accuracy of our learned solutions, we apply our single-volume algorithm for 11 discrete material volume constraints sampled uniformly across the target range. As can be seen in Figure 4.5, our solution space network does not compromise the quality of individual designs. The mean and maximum errors in compliance and volume violation are 0.75%, 3.83%, 0.5% and 2.83%, respectively. We therefore conclude that our model successfully learned the continuous solution space. Furthermore, we argue that the level of accuracy is acceptable for design exploration in many engineering applications.

Different solution spaces. To further analyze the behavior of our solution space approach, we conducted two additional experiments: the beam example with fixed volume but varying location for the applied forces and the bridge example with varying density constraint; see

Figure 4.4. Both examples confirmed our initial observations, showing smoothly evolving topology and compliance values close to the single-solution reference. For both examples, we sample 25 volume fractions/force locations during each iteration using stratified sampling. In addition to our single solution setup, we shuffle the density pairs randomly during the MMSE fitting step. The training takes 280 iterations in total, leading to 37.3 hours. Once trained, the inference enables real-time exploration of the solution space (0.014s/71.4FPS for 300×100 samples).

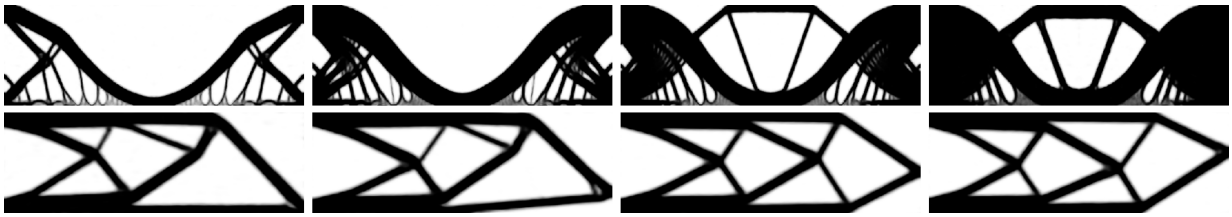


Figure 4.4 – Different solution spaces. Here we show additional results varying the volume fraction constraints for the bridge example (first row) as well as varying the applied force location for the cantilever beam (second row). See Figure 4.3 for boundary conditions.

Comparison with supervised setting. We further compare our self-supervised training techniques with a supervised setting under the same computational budget. The cost of performing 1,000 optimization steps ($5.25h$) with our larger solution space network is similar to (but somewhat lower than) the cost of computing 11 solutions under single volume constraints. We select 11 volume constraint values uniformly and train the network from these solutions in a supervised fashion. As can be seen in Figure 4.6, the network performs poorly for unseen data leading to infeasible designs and significant volume violations. On the contrary, our self-supervised approach leads to physically valid material distributions.

4.5.3. Irregular BCs and 3D Results

Our formulation extends to more complex boundaries, which we illustrate on a set of additional examples. Figure 4.7 shows multiple examples where densities are constrained on circular sub-domains and the Dirichlet boundary $\partial\Omega_D$ is also circular in one of the two examples. Our method shows promise in 3D, as demonstrated on the two examples shown in Figure 4.8. Due to the symmetry of the configuration, we apply symmetric boundary conditions to reduce the domain of interests to half and quarter for the cantilever beam and bridge example respectively to save computational cost and memory usage. Our method finds smooth solutions with various supporting features for the two tasks.

4.5.4. Ablation Studies

Here we provide evidence for the necessity of using a sensitivity filter, our moving mean squared error loss during the optimization process, and the influence of different neural

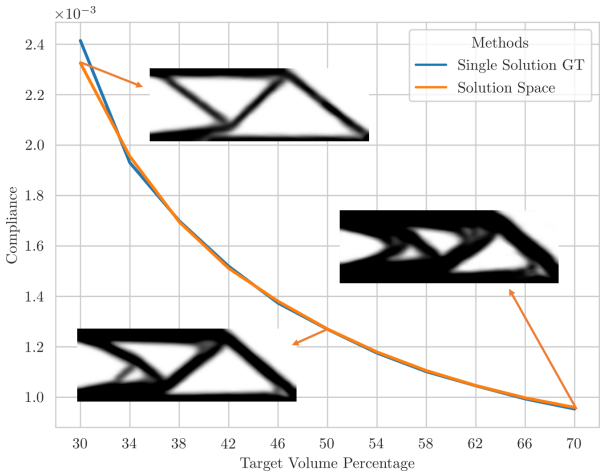


Figure 4.5 – Results and compliance comparisons between evaluating our solution space network at discrete volume locations and the reference solution produced by our method in the single volume constraint case. As can be seen, the learned solution space does not compromise the quality of individual solutions.

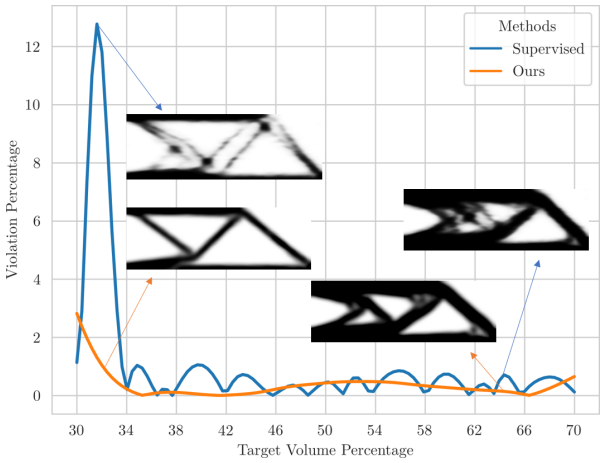


Figure 4.6 – Comparisons with supervised setting. Our self-supervised learning methods outperforms its supervised counterparts with less computational cost. The target volume constraint is shown on the x-axis and the constraint violation (in percentage) of the resulting structures is given on the y-axis.

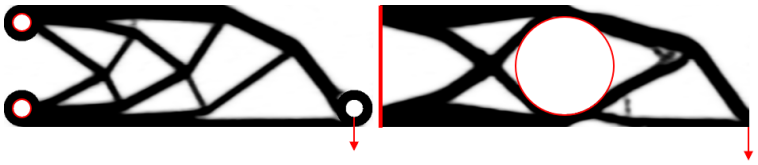


Figure 4.7 – Curved boundaries: In the left example 3 holes have been put in the design using constraints on the density field. On the right, the density field is constrained to have a hole in the middle.

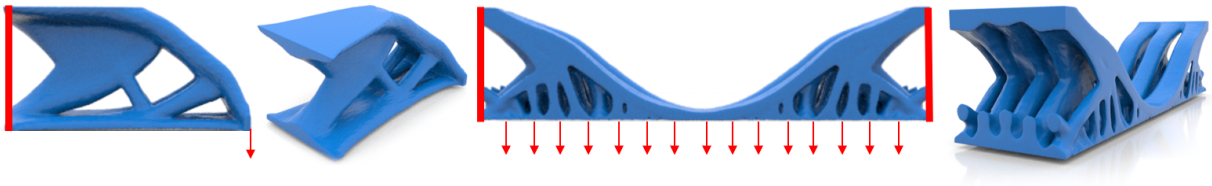


Figure 4.8 – The solutions for two 3D examples demonstrate the promise of our method in 3D. A 3D cantilever beam (left) and a 3D bridge (right). The mesh has been generated using a marching-cubes algorithm [Lewiner et al., 2003].

networks. In the first test, we do not rely on the optimality criterion method to compute the target density field nor do we use the mean squared error. Rather, we add a soft penalty term to satisfy the volume constraint and update the neural network only once per iteration. In the second test, we adopt the proposed method without filtering. Comparing with our reference solutions at different iterations, the alternatives either converge significantly slower

or arrive at poor local minima with many artefacts, such as disconnected struts or rough boundaries. We further compare our network structure with a ReLU-MLP as proposed in TOuNN [Chandrasekhar and Suresh, 2020]. As can be seen from Figure 4.10, this approach fails to capture much of the geometric features that our method (and SIMP) produce, leading to more compliant (i.e., less optimal) designs (compliance: 0.001196). The Fourier feature network [Tancik et al., 2020] leads to comparable results (compliance: 0.001172) and is thus also a valid option for our method.

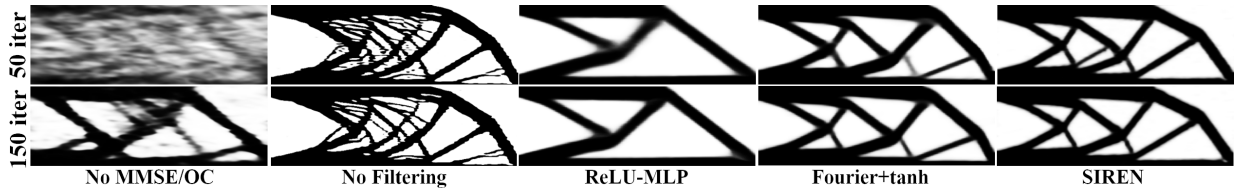


Figure 4.9 – Ablation studies on the beam example. $1st$ column: without moving mean squared error and optimality criterion—the density-space gradient is directly applied to the network, which is updated once per optimization iteration. $2nd$ column: proposed method without filtering. $3rd$ column: proposed method using ReLU-MLP as the density network, SIREN is adopted as simulation network to obtain accurate equilibrium displacement. $4th$ column: Proposed method with Fourier feature network. $5th$ column: Proposed method with SIREN network. We verify that our moving mean squared error, filtering, and choice of activation function are all quintessential.

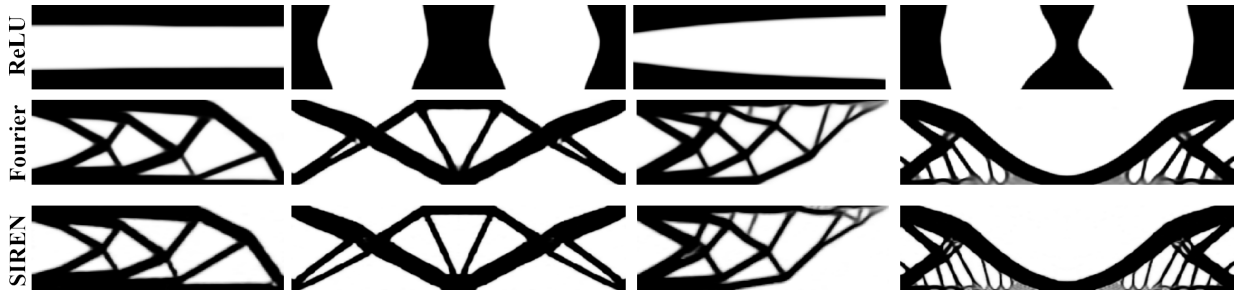


Figure 4.10 – We use different network architectures to parameterize both the density and displacement field. Since the ReLU-MLP fails to capture the high-frequency details, its solution for the forward problem is far from being accurate, resulting in meaningless structures for the inverse problems. Fourier feature and SIREN networks produced similar results, thus, suitable for our method.

4.5.5. Training Details.

We use Adam [Kingma and Ba, 2014] as our optimizer for both displacement and density networks and the learning rate of both is set to be $3 \cdot 10^{-4}$ for all experiments. We use $\omega_0 = 60$ for the first layer and 60 neurons in each hidden layer in 2D, and 180 hidden neurons in 3D. For the solution space learning setup, we use 256 neurons in each hidden layer in the

density network to represent the larger solution space. For all experiments, we initialize the output of the density network close to a uniform density distribution of the target volume constraint by initializing the weights of the last layer close to zero and adjusting the bias accordingly. We used $E_1 = 1, \nu = 0.3, p = 3, n_b = 50$ and $[n_x, n_y] = [150, 50]$ in 2D and $[n_x, n_y, n_z] = [80, 40, 20]$ in 3D. The training iterations for the 3D examples are 100 with a per iteration cost of $131.7s$. All timings in the paper are reported on a GeForce RTX 2060 SUPER graphics card.

4.6. Conclusion

We propose a novel, fully mesh-free TO method using implicit neural representations. At the heart of our method lie two neural networks that represent force-equilibrium configurations and optimal material distribution, respectively. Experiments demonstrate that our solutions are qualitatively on par with the standard FEM method for structural compliance minimization problems, yet further enables self-supervised learning of solution spaces. The proposed method can handle irregular boundary conditions due to its mesh-free nature and is applicable to 3D problems as well. As such we consider it a steppingstone towards solving many varieties of inverse design problems using neural networks.

Limitations and Future Works. As we adopted the sigmoid function to enforce box constraints, it naturally leads to small gradients when approaching 0 or 1. Although it did not lead to convergence issue for us in practice, better ways of enforcing box constraints in density space is an interesting avenue for future work. Like for other Monte Carlo-based methods, advanced sampling strategies, *e.g.*, importance sampling can also be explored to speed up the optimization process.

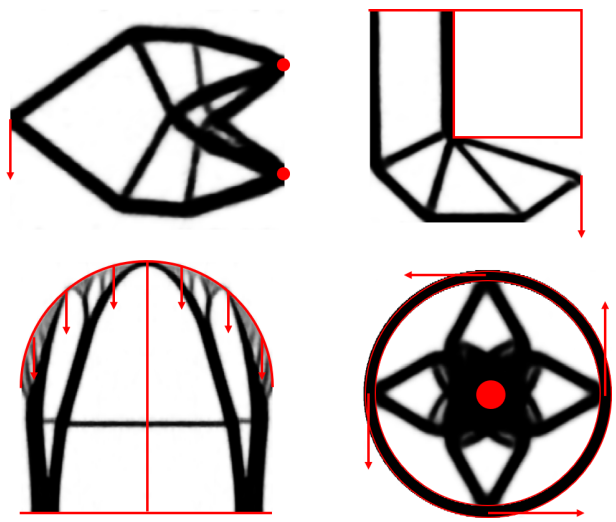
Acknowledgements

This research was supported by the Discovery Accelerator Awards program of the Natural Sciences and Engineering Research Council of Canada (NSERC), the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (Grant No. 866480) and the Swiss National Science Foundation (Grant No. 200021_200644).

4.S. Supplementary Material

4.S.1. Additional Results Obtained with our Method

We show more results obtained using our mesh-free topology optimization approach.



Top-left: Two point locations are fixed on the right with a point-wise force applied on the left. Top-right The structure is held on the top and a point-wise force is applied on the bottom right. The top right of the density field is constrained to be empty. Bottom-left: The structure is held on the bottom and a distributed force field is applied across a half-circle. Bottom-right: The structure is held in the middle and four tangential forces are applied on an outer annulus of material. The target volume ratios in the same order are 30%, 20%, 25% and 40%.

4.S.2. Sensitivity Analysis.

Here we provide the derivation of the total gradient $d\mathcal{L}_{\text{comp}}/d\theta$ for the density network weights θ, taking into account that when we change θ the displacement u changes, such that it stays in static equilibrium.

First, we would like to note that there are two different perspectives on the optimization problem in Eq. (4.3.1). The difference in formulations is due to the choice of

(1) eliminating constraints and keeping just one set of variables (ρ), in this case $u = Simulation(\rho)$

(2) using separate variables, ρ and u, and introducing explicit equilibrium constraints $\nabla\mathcal{L}_{\text{sim}} = 0$

Formulation (1) is used in the main paper, as it is in our opinion more easily readable. Formulation (2) corresponds to the standard formulation of constrained optimization problems

using Lagrange multipliers. In the Lagrangian L, all arguments (u, ρ and the Lagrange multipliers) are independent and are only coupled through the solution condition $\nabla L = 0$. Here, the derivation of the gradient will rely on formulation (2).

One way of computing the total gradient of the compliance objective is to solve for the discrete adjoint variables. Solving the discretized adjoint problem takes the form of

$$\frac{\partial^2 \mathcal{L}_{\text{sim}}}{\partial \gamma^2} \lambda = -\frac{\partial \mathcal{L}_{\text{comp}}}{\partial \gamma} \tag{4.S.1}$$

however this is a very large dense linear system which would be costly to solve, instead we rely on a point-wise argument to derive the adjoint gradient. We apply the chain rule with the neural network to get the total gradient

$$\frac{d\mathcal{L}_{\text{comp}}}{d\theta} = \int_\Omega \frac{de(\omega)}{d\rho(\omega)} \frac{\partial \rho(\omega)}{\partial \theta} \, d\omega \tag{4.S.2}$$

where $de(\omega)/d\rho(\omega)$ is the point-wise total gradient of the point-wise compliance e. To derive this point-wise gradient we start by introducing the Lagrangian

$$L(u, \rho, \lambda, \mu) = \int_\Omega e + \lambda^T (\nabla \cdot \sigma(u) - f) \, d\omega + \int_{\partial\Omega} \mu^T (\sigma(u) n) \, d\omega \tag{4.S.3}$$

$$= \int_\Omega e - \nabla \lambda : \sigma(u) \, d\omega - \lambda^T f \, d\omega + \int_{\partial\Omega} -\lambda^T (\sigma(u) n) + \mu^T (\sigma(u) n) \, d\omega \tag{4.S.4}$$

$$= \int_\Omega e - \epsilon(\lambda) : C : \epsilon(u) - \lambda^T f \, d\omega + \int_{\partial\Omega} (\mu - \lambda)^T (\sigma(u) n) \, d\omega \tag{4.S.5}$$

where we used integration by parts and leveraged the fact that $\nabla \lambda : \sigma = \epsilon(\lambda) : \sigma$ due to the symmetry of the stress tensor σ. C is the stiffness tensor in Hooke's law, that is, $\sigma = C : \epsilon$ and $\sigma_{ij} = \sum_{kl} C_{ijkl} \epsilon_{kl}$.

In the case of topology optimization for linear elasticity the adjoint variables are readily available through $\lambda = \mu = u$ [Allaire et al., 2004; Luo et al., 2005]. Here we briefly show the derivation:

We denote the directional derivative of F w.r.t.x in the direction of h as

$$D_{x,h} F(x) = \lim_{t \to 0} \frac{F(x + th) - F(x)}{t} \tag{4.S.6}$$

with which the adjoint variables are defined by [Tröltzsch, 2010]

$$\forall h \in (H^1)^2 : \quad D_{u,h} L(u, \rho, \lambda, \mu) = 0 \, . \tag{4.S.7}$$

and by applying the definition of the directional derivative, this leads to

$$\forall h \in (H^1)^2 : \quad D_{u,h} L(u, \rho, \lambda, \mu) = \int \epsilon(h) : C : \epsilon(u) - \epsilon(h) : C : \epsilon(\lambda) \, d\omega = 0 \tag{4.S.8}$$

which is true for $\lambda = u$.

We can then plugin the adjoint variables into the Lagrangian to get the point-wise gradient of the compliance with respect to the density as

$$\forall \delta\rho \in H^1 : D_{\rho,\delta\rho} L(u, \rho, \lambda, \mu) = \int_\Omega -\delta\rho \, \frac{1}{2}\epsilon(u) : \frac{\partial C}{\partial \rho} : \epsilon(u) \, d\omega \tag{4.S.9}$$

Since this holds for all functions $\delta\rho$ in H^1 we can conclude that the point-wise gradient is

$$\frac{de(\omega)}{d\rho(\omega)} := -\frac{1}{2}\epsilon(\omega) : \frac{\partial C(\omega)}{\partial \rho(\omega)} : \epsilon(\omega) = -\frac{\partial e(\omega)}{\partial \rho(\omega)} \tag{4.S.10}$$

which notably has the opposite sign of the partial gradient $\partial e / \partial \rho$ where the implicit change in the displacement u is not taken into account.

Batch density-space gradient. By sampling a batch of ω^b and $\rho^b = \rho(\omega^b)$ we can approximate the integrals and get a discrete version of the loss

$$\mathcal{L}_{\text{comp}} = \int_\Omega e \, d\omega \approx \mathcal{L}'_{\text{comp}} = \frac{|\Omega|}{n} \sum_i^n e(\omega_i^b) \tag{4.S.11}$$

using the derivations above we arrive at

$$\frac{d\mathcal{L}_{\text{comp}}}{d\theta} = \int_\Omega \frac{de(\omega)}{d\rho(\omega)} \frac{\partial \rho(\omega)}{\partial \theta} \, d\omega = -\int_\Omega \frac{\partial e(\omega)}{\partial \rho(\omega)} \frac{\partial \rho(\omega)}{\partial \theta} \, d\omega \tag{4.S.12}$$

$$\approx -\frac{|\Omega|}{n} \sum_i^n \frac{\partial e(\omega_i^b)}{\partial \rho(\omega_i^b)} \frac{\partial \rho(\omega_i^b)}{\partial \theta} = -\frac{\partial \mathcal{L}'_{\text{comp}}}{\partial \rho^b} \frac{\partial \rho^b}{\partial \theta} \tag{4.S.13}$$

$$= \frac{d\mathcal{L}'_{\text{comp}}}{d\rho^b} \frac{\partial \rho^b}{\partial \theta}. \tag{4.S.14}$$

4.S.3. Additional Information for the Results

Here we provide more detailed descriptions of the results in the main paper. See the following table for the domain sizes and boundary condition enforcements.

	Ω	$\hat{V}$	$d_x(\omega)$	$d_y(\omega)$	$d_z(\omega)$
Short Beam	1.5×0.5	0.5	ω_x	ω_x	—
Distributed	1.5×0.5	0.4	ω_x	ω_x	—
Long Beam	1.0×0.5	0.4	$\omega_x(\omega_x - 1.0)$	ω_x	—
Bridge	1.0×0.5	0.4	$\omega_x(\omega_x - 1.0)$	ω_x	—
3D Beam	$1.0 \times 0.5 \times 0.25$	0.2	ω_x	ω_x	$\omega_x(\omega_z - 0.25)$
3D Bridge	$1.0 \times 0.5 \times 0.25$	0.2	$\omega_x(\omega_x - 1.0)$	ω_x	$\omega_x(\omega_z - 0.25)$

We apply a concentrated force at the bottom right corner for the *Short Beam* example and distributed forces on the top boundary for the *Distributed* example, all along the negative y-axis. A concentrated force is applied at the bottom right corner of the new design domain

for the *Long Beam example* and distributed loading forces are added to the bottom boundary for the *Bridge* example.

The design domains for both the *Long Beam* and the *Bridge* are 2.0×0.5, however, due to the symmetric configuration, we only perform simulation and optimization on half of the domain to avoid unnecessary computations. The domains are then 1.0×0.5. For visualization, we mirror the other half. We use the term $\omega_x(\omega_x - 1)$ in $d_x(\omega)$, this enforces the left boundary to have no displacement in x direction and the normal displacements of the right boundary to vanish, similar constructions are used in the other examples.

For the $3D$ *Beam* example, we run our method only on half of the design domain along z-axis, due to symmetry and the actual design domain is $1.0 \times 0.5 \times 0.25$. Distributed forces are applied at $x = 1.0, y = 0$ and $z \in [0.0, 0.25]$.

For the $3D$ *Bridge* example our method is only performed on a quarter of the design domain $2.0 \times 0.5 \times 0.5$, where both x and z dimension are reduced to half. Forces are applied to the bottom plane where $y = 0, x \in [0.0, 1.0]$ and $z \in [0.0, 0.25]$. All forces applied are along the negative y-axis.

4.S.3.1. Curved Boundaries. We have used two additional types of constraints in these examples, constraining densities and constraining displacements on curved boundary. Since derivatives of the density field $\rho(\omega)$ do no appear in the objectives directly, constraining densities is rather straight forward. We just overwrite the density when the point ω falls into a constraint region Ω_ρ^c:

$$\tilde{\rho}(\omega) = \begin{cases} \rho(\omega) & \text{if } \omega \notin \Omega_\rho^c \\ \rho^c(\omega) & \text{if } \omega \in \Omega_\rho^c \end{cases} \tag{4.S.15}$$

and use the resulting field $\tilde{\rho}$ in place of ρ. For the displacement u, applying constraints is more difficult since its derivatives appear in the objectives, requiring that the field u is sufficiently smooth inside the domain. We use a length factor d [McFall and Mahan, 2009], to define the constrained displacement field

$$u(\omega) = d(\omega)\Phi_u(\omega) \tag{4.S.16}$$

where d has to be zero on the boundary and have non-zero first-order derivatives on the boundary (otherwise the first-order derivatives are also constrained to zero on the boundary, which we do not want). Here we show a simple analytic method for defining the length factor and is accurate on the whole primitives not only on sampled points: we allow for n primitives that require the ability to compute a C^1 continuous distance function, giving $d_1^2, \ldots d_n^2$ squared distances. We then combine them to construct a smooth length factor

$$d(\omega) = \sqrt{\text{mix}(d_1^2, \ldots, d_n^2)} \tag{4.S.17}$$

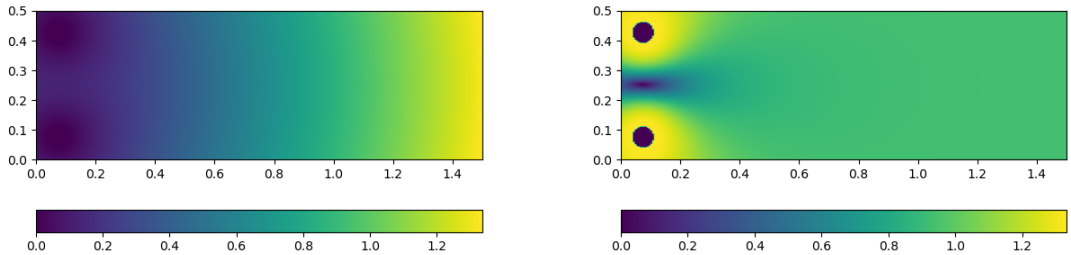


Figure 4.11 – Left: Length factor $d(\omega)$ of the curved boundary example. Right: Norm of gradient of length factor, $\|\partial d/\partial \omega\|$

where mix repeatedly merges two distances by applying a function $m(\cdot,\cdot)$ in a tree like fashion, where we use the following function

$$m(d_1^2, d_2^2) = \frac{d_1^2 d_2^2}{d_1^2 + d_2^2 + \varepsilon} \qquad \varepsilon = 10^{-4} \tag{4.S.18}$$

This function is basically the power smooth min [Quilez] with $k = 2$ plus ε, notably this function is zero when either distance d_1 or distance d_2 is zero and seems to have nice first-order derivative properties.

Data of curved boundary examples. The domain for both examples is 1.5×0.5.

Three-hole-design: The bottom left of the domain is [0,0]. The holes have inner radius 0.035 and outer radius 0.075, they are located at {[0.075, 0.425], [0.075, 0.075], [1.425, 0.075]}. The force is applied at $[1.425, 0.04]$. The length factor of one circle was defined using

$$d(\omega) = \begin{cases} 0 & \text{if } \|c - \omega\| < r \\ \|c - \omega\| - r & \text{otherwise} \end{cases} \tag{4.S.19}$$

where c is the center and r is the radius, and then the length factors of two circles were combined using the procedure explained above.

Hole in the middle-design: The radius of the circle is 0.2 and the center of the circle is at the center of the domain.

Chapter 5

Conclusion

We conclude by summarizing the hypotheses of the articles and by presenting ideas for future directions of research.

This thesis addresses the question whether we can create better computational tools by leveraging problem-specific structure. We have presented new methods in multiple different domains that both simulate and optimize the corresponding physical system in a more effective way. The first two articles present methods which empower users to get better results faster, while the third article allows for an effective exploration of the solution space.

In the first article, Chapter 2, we have looked at the energy dissipation in the simulation of incompressible fluids. We aimed at reducing the dissipation as it has a noticeable visual impact on the end result. We presented a new time integration method for incompressible fluids based on reflections, effectively reducing the amount of dissipation due to the constraint projection. This change is simple to apply to existing code. The resulting animations are more dynamic. Our method does not have higher order accuracy, but since the publication, we [Narain et al., 2019] have improved the method to be second order accurate. The resulting method is effective both in terms of computational speed and reducing the energy dissipation.

In Chapter 3, we presented how to apply Gauss-Newton effectively in the sensitivity analysis context. We use the structure of the non-linear least squares objective to find a search direction, which is always descending. Additionally, by algebraically reformulating the search direction computation, we presented a more efficient way to compute the search direction for a class of problems with certain sparsity patterns. We applied the method to multiple problems including inverse elastic design, shell design and optimal control. The results are promising and the connection we make between sensitivity analysis and constrained optimization opens the path to efficient computation of second order sensitivities. Our method is easily understandable and easily interpretable.

Finally, in the third article, Chapter 4, we have focused on the optimization of material distributions, generating minimally compliant designs under load. Our method pushes the

boundaries of what is possible using neural networks by showcasing inverse design only using implicit representations without relying on meshes. Our tests showed that a naive implementation of this concept turns out to be not competitive, however we improved the results over the baseline approach by guiding the neural network during the optimization to ensure convergence to a desirable solution. Additionally, we adapted multiple approaches from the mesh-based to the mesh free paradigm to improve robustness and speed. We also showed that we can learn whole spaces of solutions using our method, allowing users to easily examine different solutions in real-time.

5.1. Research directions

In the first article we have presented a numerical method to decrease energy dissipation in the simulation of incompressible fluids. Part of the goal was to improve the visual look, but measuring whether the visual look is better is difficult, since audiences' visual taste arises in a complex fashion. Um et al. [2017] did a perceptual study of liquids and it would be interesting to see whether one can extend this and learn a perceptual metric from human data. A machine based estimate of how good a fluid simulation looks would allow to automatically tune a simulation model according to the perceptual metric.

An alternative path for compelling results may be derived from the idea of adding more constraints related to vorticity. Along these lines Dinev et al. [2018] presented a projection method for elastic bodies which both can improve energy preservation and robustness against numerical "explosions". A similar approach for other phenomena may produce very desirable results.

In Chapter 3, we have looked at accelerating sensitivity analysis based optimization for problems for which sparse direct solvers are efficient. However, for problem classes for which this is not the case, iterative solvers can be very efficient (see e.g. [Yan et al., 2018]) when the problem is not numerically stiff. But the case of difficult sparsity patterns and numerical stiffness often arises in inverse design problems and so far seem to have eluded researchers. A solution to this problem class would be widely applicable.

While sparse Gauss-Newton relies on the fact that the objective has a non-linear least squares form, it may be worthwhile to look into using other structures at hand, such as separability of objectives, collections of objectives which on their own are easy to optimize but as a sum are difficult to optimize (see ADMM), smoothness of the solution or element hierarchies.

We have computed the search direction using the Gauss-Newton approximation. In general, it is an unsolved problem what the optimal convex approximation to the objective is. Majorization-minimization is another method of computing the search direction based on the idea of putting convex bounds on the non-convex objective. In practice, it would very useful

if one could automatically find a very good Hessian approximation in terms of convergence and/or to what kind of local minimum the algorithm converges.

Recently, large advances in machine learning have been made. This raises the question whether there are machine learning methods and techniques, which can help solve problems similar in spirit of this thesis. We ask ourselves how machine learning can be used in simulation and optimizations, and at least a few major categories of how to apply machine learning come to mind. First, and probably the most common one is prediction, i.e. computing guesses for initializations and storing solutions spaces. One obvious application is predicting initializations to reduce overall computational cost. In practice, the more impact-full research direction may be to look into computing guesses which are closer to global optima, which are in general especially difficult to find. Neural networks or other machine learning based techniques might help encoding heuristics and expertise. Additionally, one could try to develop design tools which are based on natural language or intent, which seemed impossible just a few decades ago. Second, machine learning based techniques have led to much progress in stylization, as they seem to be able to capture visual similarity much better than previous methods. This could be directly applied to inverse design to measure style-alikeness and to create more intuitive design tools involving stylization. Third, machine learning can be used to discover structure in data. In our experience applying machine learning based techniques, such as non-linear dimensionality reduction, to simulation is very difficult as many machine learning techniques have a lot of overhead compared to analytic methods. In practice, it seems advantageous to capture as much as possible at once of the simulation to reduce overhead and it is more challenging for instances where we look for concise models. Recently Cranmer et al. [2020] have demonstrated an interesting approach, which takes in particle trajectories and produces symbolic models for the forces between the particles. Applying similar, non-exact techniques to automatically find symbolic models may have applications in topics such as geometric multi grid, non-linear homogenization and developing new discrete elements.

In conclusion, we have presented multiple computational methods which improve on previous methods in important ways. We have discussed their advantages and disadvantages. In doing so, we hope to have created new useful tools, created better understanding and inspired future research.

Bibliography

Niels Aage, Erik Andreassen, Boyan S Lazarov, and Ole Sigmund. Giga-voxel computational morphogenesis for structural design. *Nature*, 550(7674):84–86, 2017.

Grégoire Allaire, François Jouve, and Anca-Maria Toader. Structural optimization using sensitivity analysis and a level-set method. *Journal of computational physics*, 194(1): 363–393, 2004.

Grégoire Allaire, Frédéric De Gournay, François Jouve, and Anca-Maria Toader. Structural optimization using topological and shape sensitivity via a level set method. *Control and cybernetics*, 34(1):59, 2005.

Erik Andreassen, Anders Clausen, Mattias Schevenels, Boyan S Lazarov, and Ole Sigmund. Efficient topology optimization in matlab using 88 lines of code. *Structural and Multidisciplinary Optimization*, 43(1):1–16, 2011.

Alexis Angelidis and Fabrice Neyret. Simulation of smoke based on vortex filament primitives. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '05, pages 87–96, 2005. ISBN 1-59593-198-8. doi: 10.1145/1073368.1073380. URL `http://doi.acm.org/10.1145/1073368.1073380`.

Saurabh Banga, Harsh Gehani, Sanket Bhilare, Sagar Patel, and Levent Kara. 3d topology optimization using convolutional neural networks. *arXiv preprint arXiv:1808.07440*, 2018.

Peter W Battaglia, Razvan Pascanu, Matthew Lai, Danilo Rezende, and Koray Kavukcuoglu. Interaction networks for learning about objects, relations and physics. *arXiv preprint arXiv:1612.00222*, 2016.

Christopher Batty, Florence Bertails, and Robert Bridson. A fast variational framework for accurate solid-fluid coupling. *ACM Trans. Graph.*, 26(3), July 2007. ISSN 0730-0301. doi: 10.1145/1276377.1276502. URL `http://doi.acm.org/10.1145/1276377.1276502`.

Martin P Bendsøe. Optimal shape design as a material distribution problem. *Structural optimization*, 1(4):193–202, 1989.

Martin P Bendsøe and Ole Sigmund. *Optimization of structural topology, shape, and material*, volume 414. Springer, 1995.

Martin P Bendsøe and Ole Sigmund. Material interpolation schemes in topology optimization. *Archive of applied mechanics*, 69(9-10):635–654, 1999.

Martin Philip Bendsoe and Noboru Kikuchi. Generating optimal topologies in structural design using a homogenization method. 1988.

Martin Philip Bendsoe and Ole Sigmund. *Topology optimization: theory, methods, and applications*. Springer Science & Business Media, 2013.

Michele Benzi, Gene H Golub, and Jörg Liesen. Numerical solution of saddle point problems. *Acta numerica*, 14:1, 2005.

Miklós Bergou, Basile Audoly, Etienne Vouga, Max Wardetzky, and Eitan Grinspun. Discrete viscous threads. *ACM Trans. Graph.*, 29(4):116:1–116:10, July 2010. ISSN 0730-0301. doi: 10.1145/1778765.1778853. URL `http://doi.acm.org/10.1145/1778765.1778853`.

Gaurav Bharaj, David I. W. Levin, James Tompkin, Yun Fei, Hanspeter Pfister, Wojciech Matusik, and Changxi Zheng. Computational design of metallophone contact sounds. *ACM Trans. Graph.*, 34(6):223:1–223:13, October 2015. ISSN 0730-0301. doi: 10.1145/2816795.2818108. URL `http://doi.acm.org/10.1145/2816795.2818108`.

Kai-Uwe Bletzinger, Matthias Firl, Johannes Linhard, and Roland Wüchner. Optimal shapes of mechanically motivated surfaces. *Computer methods in applied mechanics and engineering*, 199(5-8):324–333, 2010.

Blaise Bourdin. Filters in topology optimization. *International journal for numerical methods in engineering*, 50(9):2143–2158, 2001.

Aaditya Chandrasekhar and Krishnan Suresh. Tounn: Topology optimization using neural networks. *Structural and Multidisciplinary Optimization*, pages 1–15, 2020.

Xiang Chen, Changxi Zheng, Weiwei Xu, and Kun Zhou. An asymptotic numerical method for inverse elastic shape design. *ACM Trans. Graph.*, 33(4):95:1–95:11, July 2014. ISSN 0730-0301. doi: 10.1145/2601097.2601189. URL `http://doi.acm.org/10.1145/2601097.2601189`.

Albert Chern, Felix Knöppel, Ulrich Pinkall, Peter Schröder, and Steffen Weißmann. Schrödinger's smoke. *ACM Trans. Graph.*, 35(4):77:1–77:13, July 2016. ISSN 0730-0301. doi: 10.1145/2897824.2925868. URL `http://doi.acm.org/10.1145/2897824.2925868`.

Alexandre Joel Chorin. Numerical solution of the navier-stokes equations. *Mathematics of Computation*, 22:745–762, 1968. doi: 10.1090/S0025-5718-1968-0242392-2.

Stelian Coros, Bernhard Thomaszewski, Gioacchino Noris, Shinjiro Sueda, Moira Forberg, Robert W Sumner, Wojciech Matusik, and Bernd Bickel. Computational design of mechanical characters. *ACM Transactions on Graphics (TOG)*, 32(4):83, 2013.

Miles Cranmer, Alvaro Sanchez-Gonzalez, Peter Battaglia, Rui Xu, Kyle Cranmer, David Spergel, and Shirley Ho. Discovering symbolic models from deep learning with inductive biases. *arXiv preprint arXiv:2006.11287*, 2020.

Juan Carlos De los Reyes. *Numerical PDE-Constrained Optimization*. Springer, 2015.

Dimitar Dinev, Tiantian Liu, Jing Li, Bernhard Thomaszewski, and Ladislav Kavan. Fepr: Fast energy projection for real-time simulation of deformable objects. *ACM Transactions*

on Graphics (TOG), 37(4):1–12, 2018.

MWMG Dissanayake and Nhan Phan-Thien. Neural-network-based approximations for solving partial differential equations. *communications in Numerical Methods in Engineering*, 10(3): 195–201, 1994.

Sharif Elcott, Yiying Tong, Eva Kanso, Peter Schröder, and Mathieu Desbrun. Stable, circulation-preserving, simplicial fluids. *ACM Trans. Graph.*, 26(1), January 2007. ISSN 0730-0301. doi: 10.1145/1189762.1189766. URL `http://doi.acm.org/10.1145/1189762.1189766`.

Ronald Fedkiw, Jos Stam, and Henrik Wann Jensen. Visual simulation of smoke. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, pages 15–22, 2001. ISBN 1-58113-374-X. doi: 10.1145/383259.383260. URL `http://doi.acm.org/10.1145/383259.383260`.

Nick Foster and Dimitri Metaxas. Realistic animation of liquids. *Graphical Models and Image Processing*, 58(5):471 – 483, September 1996. ISSN 1077-3169. doi: https://doi.org/10.1006/gmip.1996.0039. URL `http://www.sciencedirect.com/science/article/pii/S1077316996900398`.

Damien Gauge, Stelian Coros, Sandro Mani, and Bernhard Thomaszewski. Interactive design of modular tensegrity characters. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 131–138. Eurographics Association, 2014.

Moritz Geilinger, David Hahn, Jonas Zehnder, Moritz Bächer, Bernhard Thomaszewski, and Stelian Coros. Add: Analytically differentiable dynamics for multi-body systems with frictional contact. *ACM Transactions on Graphics (TOG)*, 39(6), 2020.

Yotam Gingold, Adrian Secord, Jefferson Y Han, Eitan Grinspun, and Denis Zorin. A discrete model for inelastic deformation of thin shells. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2004.

Rony Goldenthal, David Harmon, Raanan Fattal, Michel Bercovier, and Eitan Grinspun. Efficient simulation of inextensible cloth. *ACM Trans. Graph.*, 26(3), July 2007. ISSN 0730-0301. doi: 10.1145/1276377.1276438. URL `http://doi.acm.org/10.1145/1276377.1276438`.

Eitan Grinspun, Anil N. Hirani, Mathieu Desbrun, and Peter Schröder. Discrete shells. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '03, pages 62–67, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association. ISBN 1-58113-659-5. URL `http://dl.acm.org/citation.cfm?id=846276.846284`.

Radek Grzeszczuk, Demetri Terzopoulos, and Geoffrey Hinton. Neuroanimator: Fast neural network emulation and control of physics-based models. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 9–20, 1998.

Xu Guo, Weisheng Zhang, and Wenliang Zhong. Doing topology optimization explicitly and geometrically—a new moving morphable components based framework. *Journal of Applied*

Mechanics, 81(8), 2014.

Ruslan Guseinov, Eder Miguel, and Bernd Bickel. Curveups: Shaping objects from flat plates with tension-actuated curvature. *ACM Trans. Graph.*, 36(4), July 2017. ISSN 0730-0301. doi: 10.1145/3072959.3073709. URL `https://doi.org/10.1145/3072959.3073709`.

Ernst Hairer, Christian Lubich, and Gerhard Wanner. *Geometric Numerical Integration: Structure-Preserving Algorithms for Ordinary Differential Equations; 2nd ed.* Springer, Dordrecht, 2006. URL `https://cds.cern.ch/record/1250576`.

S-P Han and O Fujiwara. An inertia theorem for symmetric matrices and its application to nonlinear programming. *Linear algebra and its applications*, 72:47–58, 1985.

Francis H. Harlow and J. Eddie Welch. Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface. *The Physics of Fluids*, 8(12):2182–2189, 1965. doi: 10.1063/1.1761178.

Juncai He, Lin Li, Jinchao Xu, and Chunyue Zheng. Relu deep neural networks and linear finite elements. *arXiv preprint arXiv:1807.03973*, 2018.

Matthias Heinkenschloss. Projected sequential quadratic programming methods. *SIAM Journal on Optimization*, 6, 05 1996. doi: 10.1137/0806022.

Oliver Hennigh, Susheela Narasimhan, Mohammad Amin Nabian, Akshay Subramaniam, Kaustubh Tangsali, Max Rietmann, Jose del Aguila Ferrandis, Wonmin Byeon, Zhiwei Fang, and Sanjay Choudhry. Nvidia simnetˆ{TM}: an ai-accelerated multi-physics simulation framework. *arXiv preprint arXiv:2012.07938*, 2020.

Daniel Holden, Bang Chi Duong, Sayantan Datta, and Derek Nowrouzezahrai. Subspace neural physics: Fast data-driven interactive simulation. In *Proceedings of the 18th annual ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 1–12, 2019.

Philipp Holl, Vladlen Koltun, and Nils Thuerey. Learning to control pdes with differentiable physics. *arXiv preprint arXiv:2001.07457*, 2020.

Stephan Hoyer, Jascha Sohl-Dickstein, and Sam Greydanus. Neural reparameterization improves structural optimization. *arXiv preprint arXiv:1909.04240*, 2019.

Yuanming Hu, Luke Anderson, Tzu-Mao Li, Qi Sun, Nathan Carr, Jonathan Ragan-Kelley, and Frédo Durand. Difftaichi: Differentiable programming for physical simulation. *arXiv preprint arXiv:1910.00935*, 2019a.

Yuanming Hu, Jiancheng Liu, Andrew Spielberg, Joshua B Tenenbaum, William T Freeman, Jiajun Wu, Daniela Rus, and Wojciech Matusik. Chainqueen: A real-time differentiable physical simulator for soft robotics. In *2019 International conference on robotics and automation (ICRA)*, pages 6265–6271. IEEE, 2019b.

Markus Ihmsen, Jens Orthmann, Barbara Solenthaler, Andreas Kolb, and Matthias Teschner. SPH Fluids in Computer Graphics. In *Eurographics 2014 - State of the Art Reports*, 2014. doi: 10.2312/egst.20141034.

Geoffrey Irving, Craig Schroeder, and Ronald Fedkiw. Volume conserving finite element simulations of deformable models. *ACM Trans. Graph.*, 26(3), July 2007. ISSN 0730-0301. doi: 10.1145/1276377.1276394. URL `http://doi.acm.org/10.1145/1276377.1276394`.

Caigui Jiang, Chengcheng Tang, Hans-Peter Seidel, and Peter Wonka. Design and volume optimization of space structures. *ACM Trans. Graph.*, 36(4), July 2017. ISSN 0730-0301. doi: 10.1145/3072959.3073619. URL `https://doi.org/10.1145/3072959.3073619`.

Chenfanfu Jiang, Craig Schroeder, Andrew Selle, Joseph Teran, and Alexey Stomakhin. The affine particle-in-cell method. *ACM Transactions on Graphics (TOG)*, 34(4):51:1–51:10, July 2015. doi: 10.1145/2766996.

Peter H Jones, Peter Jones, and Ove Nyquist Arup. *Ove Arup: Masterbuilder of the twentieth century*. Yale University Press, 2006.

Martin Kilian, Aron Monszpart, and Niloy J. Mitra. String actuated curved folded surfaces. *ACM Transactions on Graphics*, 36(3):25:1–25:13, May 2017. ISSN 0730-0301. doi: 10.1145/3015460. URL `http://doi.acm.org/10.1145/3015460`.

ByungMoon Kim, Yingjie Liu, Ignacio Llamas, and Jarek Rossignac. Flowfixer: Using bfecc for fluid simulation. In *Proceedings of the First Eurographics Conference on Natural Phenomena*, NPH'05, pages 51–56, 2005. ISBN 3-905673-29-0. doi: 10.2312/NPH/NPH05/051-056. URL `http://dx.doi.org/10.2312/NPH/NPH05/051-056`.

ByungMoon Kim, Yingjie Liu, Ignacio Llamas, and Jarek Rossignac. Advections with significantly reduced dissipation and diffusion. *IEEE Transactions on Visualization and Computer Graphics*, 13(1):135–144, January 2007. ISSN 1077-2626. doi: 10.1109/TVCG.2007.3. URL `http://dx.doi.org/10.1109/TVCG.2007.3`.

Doyub Kim, Oh-young Song, and Hyeong-Seok Ko. A semi-lagrangian cip fluid solver without dimensional splitting. *Computer Graphics Forum*, 27(2):467–475, April 2008a. ISSN 1467-8659. doi: 10.1111/j.1467-8659.2008.01144.x. URL `http://dx.doi.org/10.1111/j.1467-8659.2008.01144.x`.

Theodore Kim, Nils Thürey, Doug James, and Markus Gross. Wavelet turbulence for fluid simulation. *ACM Trans. Graph.*, 27(3):50:1–50:6, August 2008b. ISSN 0730-0301. doi: 10.1145/1360612.1360649. URL `http://doi.acm.org/10.1145/1360612.1360649`.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Shahar Z. Kovalsky, Meirav Galun, and Yaron Lipman. Accelerated quadratic proxy for geometric optimization. *ACM Trans. Graph.*, 35(4):134:1–134:11, July 2016. ISSN 0730-0301. doi: 10.1145/2897824.2925920. URL `http://doi.acm.org/10.1145/2897824.2925920`.

Isaac E Lagaris, Aristidis Likas, and Dimitrios I Fotiadis. Artificial neural networks for solving ordinary and partial differential equations. *IEEE transactions on neural networks*, 9(5):987–1000, 1998.

Xin Lei, Chang Liu, Zongliang Du, Weisheng Zhang, and Xu Guo. Machine learning-driven real-time topology optimization under moving morphable component-based framework. *Journal of Applied Mechanics*, 86(1), 2019.

Felix Lenders, Christian Kirches, and Andreas Potschka. trlib: A vector-free implementation of the gltr method for iterative solution of the trust region problem. *Optimization Methods and Software*, 33(3):420–449, 2018.

B.P. Leonard. A stable and accurate convective modelling procedure based on quadratic upstream interpolation. *Computer Methods in Applied Mechanics and Engineering*, 19(1): 59 – 98, 1979. ISSN 0045-7825. doi: https://doi.org/10.1016/0045-7825(79)90034-3. URL `http://www.sciencedirect.com/science/article/pii/0045782579900343`.

Thomas Lewiner, Hélio Lopes, Antônio Wilson Vieira, and Geovan Tavares. Efficient implementation of marching cubes' cases with topological guarantees. *Journal of graphics tools*, 8(2):1–15, 2003.

Yue Li, Xuan Li, Minchen Li, Yixin Zhu, Bo Zhu, and Chenfanfu Jiang. Lagrangian–eulerian multidensity topology optimization with the material point method. *International Journal for Numerical Methods in Engineering*, March 2021. doi: 10.1002/nme.6668.

Junbang Liang, Ming Lin, and Vladlen Koltun. Differentiable cloth simulation for inverse problems. 2019.

Qiyin Lin, Jun Hong, Zheng Liu, Baotong Li, and Jihong Wang. Investigation into the topology optimization for conductive heat transfer based on deep learning approach. *International Communications in Heat and Mass Transfer*, 97:103–109, 2018.

Haixiang Liu, Yuanming Hu, Bo Zhu, Wojciech Matusik, and Eftychios Sifakis. Narrow-band topology optimization on a sparsely populated grid. *ACM Transactions on Graphics (TOG)*, 37(6):1–14, 2018.

Tiantian Liu, Sofien Bouaziz, and Ladislav Kavan. Quasi-newton methods for real-time simulation of hyperelastic materials. *ACM Trans. Graph.*, 36(4), May 2017. ISSN 0730-0301. doi: 10.1145/3072959.2990496. URL `http://doi.acm.org/10.1145/3072959.2990496`.

Lu Lu, Xuhui Meng, Zhiping Mao, and George E Karniadakis. Deepxde: A deep learning library for solving differential equations. *arXiv preprint arXiv:1907.04502*, 2019.

Zhen Luo, Liping Chen, Jingzhou Yang, Y Zhang, and K Abdel-Malek. Compliant mechanism design using multi-objective topology optimization scheme of continuum structures. *Structural and Multidisciplinary Optimization*, 30(2):142–154, 2005.

Zhen Luo, Michael Yu Wang, Shengyin Wang, and Peng Wei. A level set-based parameterization method for structural shape and topology optimization. *International Journal for Numerical Methods in Engineering*, 76(1):1–26, 2008.

Mickaël Ly, Romain Casati, Florence Bertails-Descoubes, Mélina Skouras, and Laurence Boissieux. Inverse elastic shell design with contact and friction. *ACM Transactions on Graphics*, 37(6), 2018.

Kevin Stanley McFall and James Robert Mahan. Artificial neural network method for solution of boundary value problems with exact satisfaction of arbitrary boundary conditions. *IEEE Transactions on Neural Networks*, 20(8):1221–1233, 2009.

Vittorio Megaro, Jonas Zehnder, Moritz Bächer, Stelian Coros, Markus Gross, and Bernhard Thomaszewski. A computational design tool for compliant mechanisms. *ACM Trans. Graph.*, 36(4):82:1–82:12, July 2017. ISSN 0730-0301. doi: 10.1145/3072959.3073636. URL `http://doi.acm.org/10.1145/3072959.3073636`.

D Metaxas and J Popovic. Weakly compressible sph for free surface flows. 2007.

Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *European Conference on Computer Vision*, pages 405–421. Springer, 2020.

Jeroen Molemaker, Jonathan M. Cohen, Sanjit Patel, and Junyong Noh. Low viscosity flow simulations for animation. In *SCA '08: Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 9–18, 2008. doi: 10.2312/SCA/SCA08/009-018.

Rajaditya Mukherjee, Longhua Wu, and Huamin Wang. Interactive two-way shape design of elastic bodies. *Proc. ACM Comput. Graph. Interact. Tech.*, 1(1):11:1–11:17, July 2018. ISSN 2577-6193. doi: 10.1145/3203196. URL `http://doi.acm.org/10.1145/3203196`.

Patrick Mullen, Keenan Crane, Dmitry Pavlov, Yiying Tong, and Mathieu Desbrun. Energy-preserving integrators for fluid animation. *ACM Trans. Graph.*, 28(3):38:1–38:8, July 2009. ISSN 0730-0301. doi: 10.1145/1531326.1531344. URL `http://doi.acm.org/10.1145/1531326.1531344`.

Przemyslaw Musialski, Christian Hafner, Florian Rist, Michael Birsak, Michael Wimmer, and Leif Kobbelt. Non-linear shape optimization using local subspace projections. *ACM Trans. Graph.*, 35(4):87:1–87:13, July 2016. ISSN 0730-0301. doi: 10.1145/2897824.2925886. URL `http://doi.acm.org/10.1145/2897824.2925886`.

Rahul Narain, Jonas Zehnder, and Bernhard Thomaszewski. A second-order advection-reflection solver. *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, 2(2):1–14, 2019.

Zhenguo Nie, Tong Lin, Haoliang Jiang, and Levent Burak Kara. Topologygan: Topology optimization using generative adversarial networks based on physical fields over the initial domain. *arXiv preprint arXiv:2003.04685*, 2020.

J. Panetta, M. Konaković-Luković, F. Isvoranu, E. Bouleau, and M. Pauly. X-shells: A new class of deployable beam structures. *ACM Trans. Graph.*, 38(4):83:1–83:15, July 2019. ISSN 0730-0301. doi: 10.1145/3306346.3323040. URL `http://doi.acm.org/10.1145/3306346.3323040`.

Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*,

pages 165–174, 2019.

Sang Il Park and Myoung Jun Kim. Vortex fluid for gaseous phenomena. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '05, pages 261–270, 2005. ISBN 1-59593-198-8. doi: 10.1145/1073368.1073406. URL `http://doi.acm.org/10.1145/1073368.1073406`.

Richard Peng and Santosh Vempala. Solving sparse linear systems faster than matrix multiplication, 2020.

Yue Peng, Bailin Deng, Juyong Zhang, Fanyu Geng, Wenjie Qin, and Ligang Liu. Anderson acceleration for geometry optimization and physics simulation. *ACM Trans. Graph.*, 37(4):42:1–42:14, July 2018. ISSN 0730-0301. doi: 10.1145/3197517.3201290. URL `http://doi.acm.org/10.1145/3197517.3201290`.

Jesús Pérez, Miguel A. Otaduy, and Bernhard Thomaszewski. Computational design and automated fabrication of kirchhoff-plateau surfaces. *ACM Trans. Graph.*, 36(4):62:1–62:12, July 2017. ISSN 0730-0301. doi: 10.1145/3072959.3073695. URL `http://doi.acm.org/10.1145/3072959.3073695`.

Tobias Pfaff, Meire Fortunato, Alvaro Sanchez-Gonzalez, and Peter W Battaglia. Learning mesh-based simulation with graph networks. *arXiv preprint arXiv:2010.03409*, 2020.

Nico Pietroni, Marco Tarini, Amir Vaxman, Daniele Panozzo, and Paolo Cignoni. Position-based tensegrity design. *ACM Trans. Graph.*, 36(6), November 2017. ISSN 0730-0301. doi: 10.1145/3130800.3130809. URL `https://doi.org/10.1145/3130800.3130809`.

Inigo Quilez. smooth minimum. `https://www.iquilezles.org/www/articles/smin/smin.htm`. Accessed: 2021-15-01.

Maziar Raissi, Paris Perdikaris, and George E Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.

Maziar Raissi, Alireza Yazdani, and George Em Karniadakis. Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations. *Science*, 367(6481):1026–1030, 2020.

Chengping Rao, Hao Sun, and Yang Liu. Physics informed deep learning for computational elastodynamics without labeled data. *arXiv preprint arXiv:2006.08472*, 2020.

André Robert. A stable numerical integration scheme for the primitive meteorological equations. *Atmosphere-Ocean*, 19(1):35–46, 1981. doi: 10.1080/07055900.1981.9649098.

Shunsuke Saito, Tomas Simon, Jason Saragih, and Hanbyul Joo. Pifuhd: Multi-level pixel-aligned implicit function for high-resolution 3d human digitization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 84–93, 2020.

Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter Battaglia. Learning to simulate complex physics with graph networks. In *International Conference on Machine Learning*, pages 8459–8468. PMLR, 2020.

Adriana Schulz, Harrison Wang, Eitan Grinspun, Justin Solomon, and Wojciech Matusik. Interactive exploration of design trade-offs. *ACM Trans. Graph.*, 37(4):131:1–131:14, July 2018. ISSN 0730-0301. doi: 10.1145/3197517.3201385. URL `http://doi.acm.org/10.1145/3197517.3201385`.

Andrew Selle, Nick Rasmussen, and Ronald Fedkiw. A vortex particle method for smoke, water and explosions. *ACM Transactions on Graphics (TOG)*, 24(3):910–914, July 2005. doi: 10.1145/1073204.1073282.

Andrew Selle, Ronald Fedkiw, Byungmoon Kim, Yingjie Liu, and Jarek Rossignac. An unconditionally stable maccormack method. *J. Sci. Comput.*, 35(2-3):350–371, June 2008. ISSN 0885-7474. doi: 10.1007/s10915-007-9166-4. URL `http://dx.doi.org/10.1007/s10915-007-9166-4`.

Ole Sigmund. A 99 line topology optimization code written in matlab. *Structural and multidisciplinary optimization*, 21(2):120–127, 2001.

Ole Sigmund. Morphology-based black and white filters for topology optimization. *Structural and Multidisciplinary Optimization*, 33(4-5):401–424, 2007.

Justin Sirignano and Konstantinos Spiliopoulos. Dgm: A deep learning algorithm for solving partial differential equations. *Journal of computational physics*, 375:1339–1364, 2018.

Vincent Sitzmann, Julien Martel, Alexander Bergman, David Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. *Advances in Neural Information Processing Systems*, 33, 2020.

Mélina Skouras, Bernhard Thomaszewski, Bernd Bickel, and Markus Gross. Computational design of rubber balloons. In *Computer Graphics Forum*, volume 31, pages 835–844. Wiley Online Library, 2012.

Mélina Skouras, Bernhard Thomaszewski, Stelian Coros, Bernd Bickel, and Markus Gross. Computational design of actuated deformable characters. *ACM Transactions on Graphics (TOG)*, 32(4):82, 2013.

Mélina Skouras, Bernhard Thomaszewski, Peter Kaufmann, Akash Garg, Bernd Bickel, Eitan Grinspun, and Markus Gross. Designing inflatable structures. 33(4), 2014.

Jos Stam. Stable fluids. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 121–128, 1999. doi: 10.1145/311535.311548.

Towaki Takikawa, Joey Litalien, Kangxue Yin, Karsten Kreis, Charles Loop, Derek Nowrouzezahrai, Alec Jacobson, Morgan McGuire, and Sanja Fidler. Neural geometric level of detail: Real-time rendering with implicit 3d shapes. *arXiv preprint arXiv:2101.10994*, 2021.

Matthew Tancik, Pratul P Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan T Barron, and Ren Ng. Fourier features let networks learn high frequency functions in low dimensional domains. *arXiv preprint arXiv:2006.10739*, 2020.

Nils Thuerey and Tobias Pfaff. Mantaflow. http://mantaflow.com, 2016.

Nils Thuerey, Theodore Kim, and Tobias Pfaff. Turbulent fluids. In *ACM SIGGRAPH 2013 Courses*, page 6. ACM, 2013.

Antonio Tomas and Pascual Martí-Montrull. Optimality of candela's concrete shells: A study of his posthumous design. *Journal of the International Association for Shell and Spatial Structures*, 51:67–77, 03 2010.

Fredi Tröltzsch. *Optimal control of partial differential equations: theory, methods, and applications*, volume 112. American Mathematical Soc., 2010.

Erva Ulu, Rusheng Zhang, and Levent Burak Kara. A data-driven investigation and estimation of optimal topologies under variable loading configurations. *Computer Methods in Biomechanics and Biomedical Engineering: Imaging & Visualization*, 4(2):61–72, 2016.

Kiwon Um, Xiangyu Hu, and Nils Thuerey. Perceptual evaluation of liquid simulation methods. *ACM Transactions on Graphics (TOG)*, 36(4):1–12, 2017.

Kiwon Um, Philipp Holl, Robert Brand, Nils Thuerey, et al. Solver-in-the-loop: Learning from differentiable physics to interact with iterative pde-solvers. *arXiv preprint arXiv:2007.00016*, 2020.

Nobuyuki Umetani, Danny M Kaufman, Takeo Igarashi, and Eitan Grinspun. Sensitive couture for interactive garment modeling and editing. *ACM Trans. Graph.*, 30(4):90, 2011.

Nobuyuki Umetani, Athina Panotopoulou, Ryan Schmidt, and Emily Whiting. Printone: Interactive resonance simulation for free-form print-wind instrument design. *ACM Trans. Graph.*, 35(6), November 2016. ISSN 0730-0301. doi: 10.1145/2980179.2980250. URL https://doi.org/10.1145/2980179.2980250.

Henk A Van der Vorst. Bi-cgstab: A fast and smoothly converging variant of bi-cg for the solution of nonsymmetric linear systems. *SIAM Journal on scientific and Statistical Computing*, 13(2):631–644, 1992.

Nico P van Dijk, Kurt Maute, Matthijs Langelaar, and Fred Van Keulen. Level-set methods for structural topology optimization: a review. *Structural and Multidisciplinary Optimization*, 48(3):437–472, 2013.

Tobias von Petersdorff. *Errors for Linear Systems*, 2020. http://terpconnect.umd.edu/~petersd/460/linsysterrn.pdf.

Etienne Vouga, Mathias Höbinger, Johannes Wallner, and Helmut Pottmann. Design of self-supporting surfaces. *ACM Trans. Graph.*, 31(4), July 2012. ISSN 0730-0301. doi: 10.1145/2185520.2185583. URL https://doi.org/10.1145/2185520.2185583.

Huamin Wang. Rule-free sewing pattern adjustment with precision and efficiency. *ACM Trans. Graph.*, 37(4):53:1–53:13, July 2018. ISSN 0730-0301. doi: 10.1145/3197517.3201320. URL http://doi.acm.org/10.1145/3197517.3201320.

Michael Yu Wang, Xiaoming Wang, and Dongming Guo. A level set method for structural topology optimization. *Computer methods in applied mechanics and engineering*, 192(1-2):

227–246, 2003.

E Weinan and Bing Yu. The deep ritz method: a deep learning-based numerical algorithm for solving variational problems. *Communications in Mathematics and Statistics*, 6(1):1–12, 2018.

Steffen Weißmann and Ulrich Pinkall. Filament-based smoke with vortex shedding and variational reconnection. *ACM Trans. Graph.*, 29(4):115:1–115:12, July 2010. ISSN 0730-0301. doi: 10.1145/1778765.1778852. URL `http://doi.acm.org/10.1145/1778765.1778852`.

Tianju Xue, Alex Beatson, Sigrid Adriaenssens, and Ryan Adams. Amortized finite element analysis for fast pde-constrained optimization. In *International Conference on Machine Learning*, pages 10638–10647. PMLR, 2020.

Guowei Yan, Wei Li, Ruigang Yang, and Huamin Wang. Inexact descent methods for elastic parameter optimization. *ACM Trans. Graph.*, 37(6):253:1–253:14, December 2018. ISSN 0730-0301. doi: 10.1145/3272127.3275021. URL `http://doi.acm.org/10.1145/3272127.3275021`.

Yonggyun Yu, Taeil Hur, Jaeho Jung, and In Gwun Jang. Deep learning for determining a near-optimal topological design without any iteration. *Structural and Multidisciplinary Optimization*, 59(3):787–799, 2019.

Jonas Zehnder, Stelian Coros, and Bernhard Thomaszewski. Designing structurally-sound ornamental curve networks. *ACM Trans. Graph.*, 35(4):99:1–99:10, 2016.

Jonas Zehnder, Espen Knoop, Moritz Bächer, and Bernhard Thomaszewski. Metasilicone: Design and fabrication of composite silicone with desired mechanical properties. *ACM Trans. Graph.*, 36(6):240:1–240:13, November 2017. ISSN 0730-0301. doi: 10.1145/3130800.3130881. URL `http://doi.acm.org/10.1145/3130800.3130881`.

Weisheng Zhang, Jian Zhang, and Xu Guo. Lagrangian description based topology optimization—a revival of shape optimization. *Journal of Applied Mechanics*, 83(4), 2016.

Weisheng Zhang, Junfu Song, Jianhua Zhou, Zongliang Du, Yichao Zhu, Zhi Sun, and Xu Guo. Topology optimization with multiple materials via moving morphable component (mmc) method. *International Journal for Numerical Methods in Engineering*, 113(11):1653–1675, 2018.

Xinxin Zhang, Robert Bridson, and Chen Greif. Restoring the missing vorticity in advection-projection fluid solvers. *ACM Trans. Graph.*, 34(4):52:1–52:8, July 2015. ISSN 0730-0301. doi: 10.1145/2766982. URL `http://doi.acm.org/10.1145/2766982`.

Yiquan Zhang, Airong Chen, Bo Peng, Xiaoyi Zhou, and Dalei Wang. A deep convolutional neural network for topology optimization with strong generalization ability. *arXiv preprint arXiv:1901.07761*, 2019.

Yongning Zhu and Robert Bridson. Animating sand as a fluid. 24:965–972, July 2005. doi: 10.1145/1073204.1073298.

Yufeng Zhu, Robert Bridson, and Danny M. Kaufman. Blended cured quasi-newton for distortion optimization. *ACM Trans. Graph.*, 37(4):40:1–40:14, July 2018. ISSN 0730-0301. doi: 10.1145/3197517.3201359. URL `http://doi.acm.org/10.1145/3197517.3201359`.

Simon Zimmermann, Roi Poranne, James M Bern, and Stelian Coros. Puppetmaster: robotic animation of marionettes. *ACM Transactions on Graphics (TOG)*, 38(4):103, 2019.