SGN: Sparse Gauss-Newton for Accelerated Sensitivity Analysis

JONAS ZEHNDER, Université de Montréal, Canada
STELIAN COROS, ETH Zürich, Switzerland
BERNHARD THOMASZEWSKI, ETH Zürich, Switzerland and Université de Montréal, Canada

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.

CCS Concepts: • **Computing methodologies → Shape modeling**; **Optimization algorithms**; • **Applied computing → Computer-aided design;**

Additional Key Words and Phrases: Sensitivity analysis, sparse Gauss-Newton, equilibrium-constrained optimization, nonlinear least-squares

ACM Reference format:
Jonas Zehnder, Stelian Coros, and Bernhard Thomaszewski. 2021. SGN: Sparse Gauss-Newton for Accelerated Sensitivity Analysis. *ACM Trans. Graph.* 41, 1, Article 4 (September 2021), 10 pages.
https://doi.org/10.1145/3470005

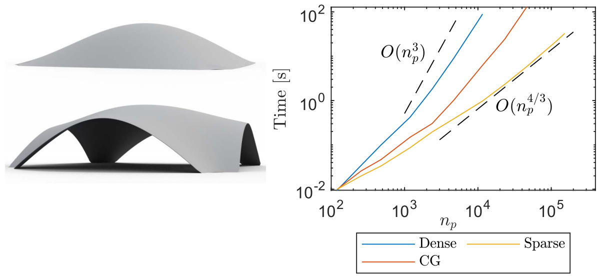

Fig. 1. Comparison between dense Gauss-Newton and our sparse formulation on a shape optimization problem for a concrete shell. *Left*: Shell roof before (*top*) and after (*bottom*) optimization. *Right*: Average timings for computing the search direction with sparse Gauss-Newton, dense Gauss-Newton, and the CG method as a function of the number of design parameters n_p.

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 but 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 article, 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.

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).
Authors' addresses: J. Zehnder, Université de Montréal, LIGUM - Department of Computer Science and Operations Research, 900 Edouard Montpetit Blvd, Montréal, Québec, Canada, H3T 1J4; email: jonas.zehnder@umontreal.ca; S. Coros, Computational Robotics Lab - Department of Computer Science, ETH Zürich, Stampfenbachstrasse 48, Zürich, Switzerland, 8092; email: scoros@inf.ethz.ch; B. Thomaszewski, Computational Robotics Lab - Department of Computer Science, ETH Zürich, Stampfenbachstrasse 48, Zürich, Switzerland, 8092 and Université de Montréal, LIGUM - Department of Computer Science and Operations Research, 2900 Edouard Montpetit Blvd, Montréal, Québec, Canada, H3T 1J4; email: bthomasz@inf.ethz.ch.

0730-0301/2021/09-ART4
https://doi.org/10.1145/3470005

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. 2012, 2014] 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 [Gauge et al. 2014; Jiang et al. 2017; 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 so 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], and material optimization [Yan et al. 2018; Zehnder et al. 2017] as well as for optimization-based forward design [Pérez et al. 2017; Umetani et al. 2011].

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 so 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 BACKGROUND

We consider constrained optimization problems of the form

$$\min_{\mathbf{x},\mathbf{p}} f(\mathbf{x},\mathbf{p}) \quad \text{s.t.} \quad \mathbf{c}(\mathbf{x},\mathbf{p}) = \mathbf{0}, \tag{1}$$

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.

3.1 Sensitivity Analysis

We exclusively consider the special but common case of problems that 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

$$\frac{d\mathbf{c}}{d\mathbf{p}} = \frac{\partial \mathbf{c}}{\partial \mathbf{p}} + \frac{\partial \mathbf{c}}{\partial \mathbf{x}}\frac{d\mathbf{x}}{d\mathbf{p}} = \mathbf{0}, \tag{2}$$

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

$$\mathrm{S} = \frac{d\mathbf{x}}{d\mathbf{p}} = -\left(\frac{\partial \mathbf{c}}{\partial \mathbf{x}}\right)^{-1}\frac{\partial \mathbf{c}}{\partial \mathbf{p}}. \tag{3}$$

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

$$\frac{df(\mathbf{x}(\mathbf{p}),\mathbf{p})}{d\mathbf{p}} = \frac{\partial f}{\partial \mathbf{p}} + \frac{\partial f}{\partial \mathbf{x}}\mathrm{S}. \tag{4}$$

We note that, by using Equation (2) in the above expression and rearranging terms, computing the gradient requires only the solution of a single linear system.

3.2 Gauss-Newton

With the gradient defined through Equation (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,

$$f(\mathbf{x},\mathbf{p}) = \sum_i \frac{w_i}{2} r_i(\mathbf{x},\mathbf{p})^2, \tag{5}$$

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

$$H = \sum_i w_i \frac{dr_i}{d\mathbf{p}}^T \frac{dr_i}{d\mathbf{p}} + \sum_i w_i r_i \frac{d^2 r_i}{d\mathbf{p}^2}, \tag{6}$$

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

$$H_{GN} = \begin{bmatrix} \frac{d\mathbf{x}}{d\mathbf{p}}^T & I \end{bmatrix} \left(\sum_i w_i \begin{bmatrix} \frac{\partial r_i}{\partial \mathbf{x}}^T \frac{\partial r_i}{\partial \mathbf{x}} & \frac{\partial r_i}{\partial \mathbf{x}}^T \frac{\partial r_i}{\partial \mathbf{p}} \\ \frac{\partial r_i}{\partial \mathbf{p}}^T \frac{\partial r_i}{\partial \mathbf{x}} & \frac{\partial r_i}{\partial \mathbf{p}}^T \frac{\partial r_i}{\partial \mathbf{p}} \end{bmatrix} \right) \begin{bmatrix} \frac{d\mathbf{x}}{d\mathbf{p}} \\ I \end{bmatrix}. \tag{7}$$

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

$$H_{GN} \cdot \delta\mathbf{p} = -\frac{df}{d\mathbf{p}}^T. \tag{8}$$

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.

4 SPARSE GAUSS-NEWTON

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

$$H_{GN} = \frac{d\mathbf{x}}{d\mathbf{p}}^T A \frac{d\mathbf{x}}{d\mathbf{p}} + B \frac{d\mathbf{x}}{d\mathbf{p}} + \frac{d\mathbf{x}}{d\mathbf{p}}^T B^T + C \tag{9}$$

with

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

Since $\frac{d\mathbf{x}}{d\mathbf{p}} = -[\frac{\partial \mathbf{c}}{\partial \mathbf{x}}]^{-1} \frac{\partial \mathbf{c}}{\partial \mathbf{p}}$, 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 $\frac{\partial \mathbf{c}}{\partial \mathbf{p}} \delta\mathbf{p} + \frac{\partial \mathbf{c}}{\partial \mathbf{x}} \delta\mathbf{x} = 0$:

$$\begin{bmatrix} \frac{d\mathbf{x}}{d\mathbf{p}}^T A + B & C + \frac{d\mathbf{x}}{d\mathbf{p}}^T B^T \\ \frac{\partial \mathbf{c}}{\partial \mathbf{x}} & \frac{\partial \mathbf{c}}{\partial \mathbf{p}} \end{bmatrix} \begin{bmatrix} \delta\mathbf{x} \\ \delta\mathbf{p} \end{bmatrix} = \begin{bmatrix} -\frac{df}{d\mathbf{p}}^T \\ 0 \end{bmatrix}. \tag{10}$$

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

$$\frac{\partial \mathbf{c}}{\partial \mathbf{x}}^T \delta\boldsymbol{\lambda} + B^T \delta\mathbf{p} + A\delta\mathbf{x} = 0, \tag{11}$$

and arrive at the extended system

$$\begin{bmatrix} A & B^T & \frac{\partial \mathbf{c}}{\partial \mathbf{x}}^T \\ B & C & \frac{\partial \mathbf{c}}{\partial \mathbf{p}}^T \\ \frac{\partial \mathbf{c}}{\partial \mathbf{x}} & \frac{\partial \mathbf{c}}{\partial \mathbf{p}} & 0 \end{bmatrix} \begin{bmatrix} \delta\mathbf{x} \\ \delta\mathbf{p} \\ \delta\boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} 0 \\ -\frac{df}{d\mathbf{p}}^T \\ 0 \end{bmatrix}. \tag{12}$$

Note that the first row enforces Equation (11), whereas the second row is obtained by using Equations (11) and (3) in the first row of Equation (10). 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 Figure 1 for an example. This result translates into dramatically improved performance for a large range of problems, as we demonstrate with our examples.

4.1 Discussion and Generalization

Relation to Sequential Quadratic Programming. System (12) 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 A.

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

$$\begin{aligned} \frac{d^2 f}{d\mathbf{p}^2} = \frac{d\mathbf{x}}{d\mathbf{p}}^T \frac{\partial^2 f}{\partial \mathbf{x}^2} \frac{d\mathbf{x}}{d\mathbf{p}} + \frac{\partial^2 f}{\partial \mathbf{x} \partial \mathbf{p}} \frac{d\mathbf{x}}{d\mathbf{p}} + \frac{d\mathbf{x}}{d\mathbf{p}}^T \frac{\partial^2 f}{\partial \mathbf{p} \partial \mathbf{x}} + \frac{\partial^2 f}{\partial \mathbf{p}^2} \\ + \sum_i \frac{\partial f}{\partial \mathbf{x}_i} \frac{d^2 \mathbf{x}_i}{d\mathbf{p}^2}. \end{aligned} \tag{13}$$

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 A. It should be noted, however, that neither Generalized Gauss-Newton nor full Newton offers 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 Section 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], and 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 Section 5.

Block Solve. If the objective f does not directly depend on the design parameters, the block structure for Equation (12) 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

$$\frac{\partial \mathbf{c}}{\partial \mathbf{p}}\delta\mathbf{p} = -\frac{\partial \mathbf{c}}{\partial \mathbf{x}}\delta\mathbf{y}\,, \quad \text{with} \quad \delta\mathbf{y} = A^{-1}\frac{\partial f}{\partial \mathbf{x}}^T . \tag{14}$$

See Appendix B 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 $\delta\mathbf{y}$. We show in Section 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.

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, and 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

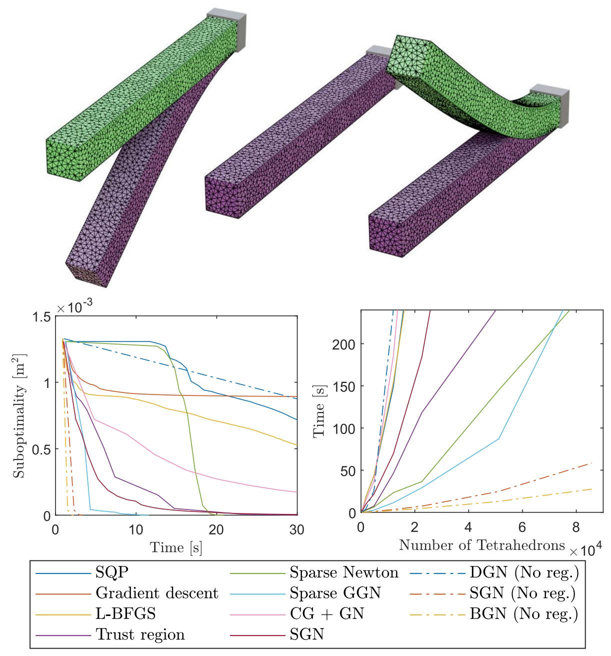

Fig. 2. Performance comparison for different solvers on an inverse elastic design problem. *Top left*: Initial rest shape (*green*) and corresponding deformed state (*purple*). *Top middle*: Target shape. *Top right*: Optimized rest shape (*green*) and corresponding deformed state (*purple*). *Bottom left*: Objective value vs. computation time for a mesh size of 3,228 vertices. *Bottom right*: Computation time vs. problem size.

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^T decomposition 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 **Generalized Gauss-Newton (GGN)** we used the inertia revealing feature of PARDISO's LDL^T decomposition 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

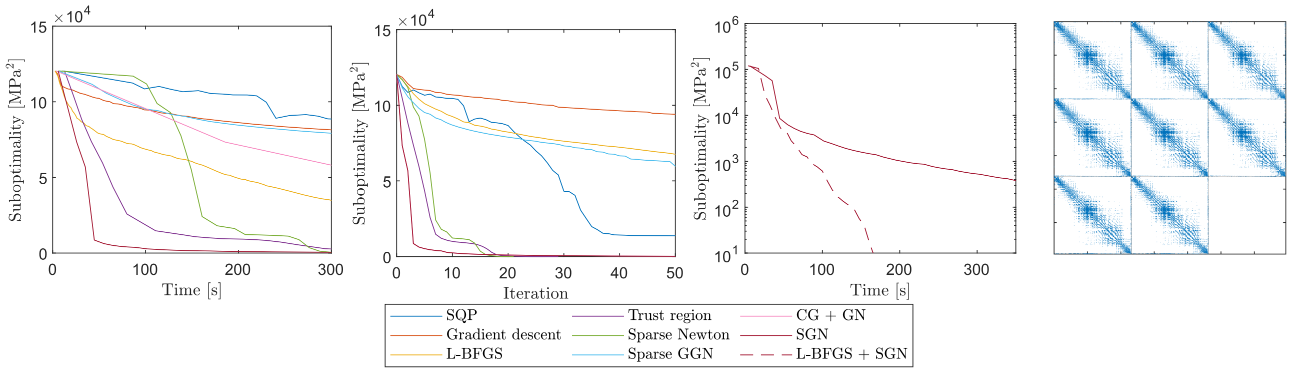

Fig. 3. Convergence of different solvers for the shell roof example with 15,443 vertices. Dense Gauss-Newton is not listed since the linear solver ran out of memory when computing the reduced Hessian. The sparsity pattern of the saddle point system, shown on the right, reveals a repetitive structure 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.

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.

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 Figure 2. The goal is to find a rest state mesh such that the resulting equilibrium state is as close as possible to a given target shape:

$$f(\mathbf{x}, \mathbf{p}) = \frac{1}{2n_x} \|\mathbf{x} - \mathbf{x}_{\text{target}}\|^2 + R(\mathbf{p}). \tag{15}$$

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 $E = 10^6 Pa$ and Poisson's ratio $\nu = 0.45$. As a 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 [Chen et al. 2014; Ly et al. 2018; Mukherjee et al. 2018], we use this example as a benchmark for evaluating the relative performance of SGN, DGN, and sparse versions of full Newton and 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 SQP using Newton's method on the KKT-conditions (see Appendix A) and an exact L_1 merit function. As a further reference point, we also compare to the **conjugate gradient (CG)** method applied directly to Equation (9) and use back-substitutions to avoid forming the dense matrix. We refer to this alternative as *CG + GN*. As a 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 Section 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 Figure 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 Figure 2 (*right*). Although the difference between BGN and SGN is small compared to the other methods, the block-solve version still offers about a 30% performance increase for lower resolutions and more than 50% for higher resolutions.

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 Figure 1. We model the roof using discrete shells [Grinspun et al. 2003] with material parameters corresponding to $E = 28GPa$ and $\nu = 0.2$ as well as a density of $2,500kg/m^3$.

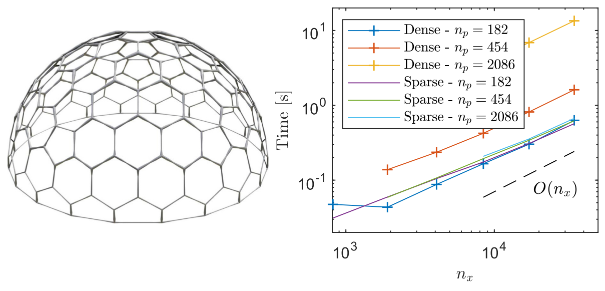

Fig. 4. Time to compute the search direction for the rod dome as a function of state size n_x. We subdivide the edges to add more state variables and compare timings for different numbers of parameters.

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

$$f(\mathbf{x},\mathbf{p}) = \sum_i ||\sigma_i(\mathbf{x},\mathbf{p})||^2 + R(\mathbf{p}), \tag{16}$$

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. As can be seen from Figure 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 Figure 3 (*right*) reveals that convergence slows down afterwards. However, the combination of SGN with L-BFGS as described in Section 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. Figure 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 Figure 1 than for the comparison shown in Figure 3, since otherwise, not all methods would converge to the same solution.

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

$$f(\mathbf{x},\mathbf{p}) = \frac{1}{2}\,||\mathbf{x}-\mathbf{x}_{\text{undef}}||^2 + \frac{1}{2}\,||\mathbf{p}||_V^2 + f_{\text{bounds}}(\mathbf{p}), \tag{17}$$

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 a 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 Figure 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 breakeven 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.

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 two examples, we consider the problem of steering a simple self-driving car so 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

$$\mathbf{c}^{t_i}(\mathbf{x},\mathbf{p}) = \mathbf{x}^{t_i} - I_{EE}(\mathbf{x}^{t_{i-1}}, \mathbf{p}^{t_i}), \tag{18}$$

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

$$f(\mathbf{x},\mathbf{p}) = \frac{w_{\text{pos}}}{2}||\mathbf{x}^N - \mathbf{x}_{\text{target}}||^2 + \frac{w_{\text{dir}}}{2}||\mathbf{d}(\mathbf{x}^N) - \mathbf{d}_{\text{target}}||^2 + R(\mathbf{p}), \tag{19}$$

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

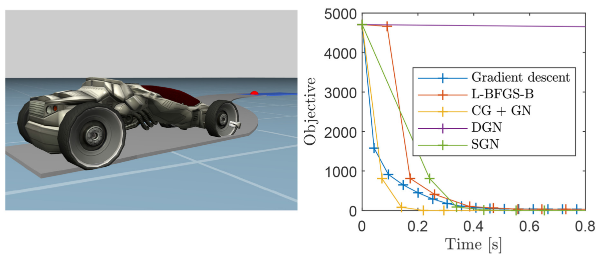

Fig. 5. Performance comparison for different solvers on the car example using 5,000 time steps.

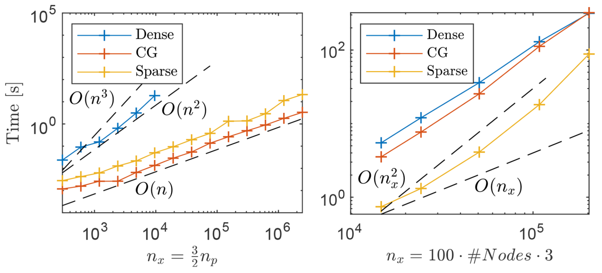

Fig. 6. Average time for computing Gauss-Newton search directions for the car (*left*) and cloth (*right*) control examples using the dense vs. our sparse Hessian and the CG method. *Left*: We increase the number of time steps, thus increasing problem size in terms of both state variable and parameters. *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.

the search direction. The measured results are shown in Figure 5 and 6.

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.

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 a 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

$$\mathbf{c}^{t_i} = \mathbf{x}^{t_i} - I_{IE}(\mathbf{x}^{t_{i-1}}, \mathbf{p}^{t_{i-1}}, \mathbf{p}^{t_i}), \tag{20}$$

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 1.66*s* of virtual time and set the step size accordingly. The

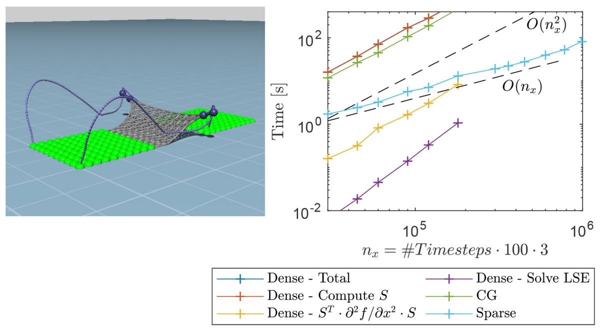

Fig. 7. Time required to compute the search direction for the cloth control problem. *Right*: We increase the number of time steps while keeping mesh resolution constant such that the problem size increases in terms of both state variables and parameters. The dense solution strategy ran out of memory for $n_x > 1.8 \cdot 10^5$.

goal of matching the target state is expressed as

$$f(\mathbf{x}, \mathbf{p}) = \frac{1}{2} \sum_{j \in \mathcal{S}} \|\mathbf{x}^j - \tilde{\mathbf{x}}^j\|^2 + R(\mathbf{p}), \tag{21}$$

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 breakdowns 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. Figure 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 Figure 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.

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.

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 breakeven 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 preconditioners for Equation (9) 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].

APPENDICES

A EQUIVALENCE RESULT

We show that, for general objectives, using the dense system obtained for second-order sensitivity analysis and the sparse system (Equation (12)) 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 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 (Equation (12)).*

PROOF. See construction in Section 4. □

We next introduce the Lagrangian for the optimization problem (Equation (1)) as

$$\mathcal{L}(\mathbf{x}, \mathbf{p}, \boldsymbol{\lambda}) = f(\mathbf{x}, \mathbf{p}) + \boldsymbol{\lambda}^T \mathbf{c}(\mathbf{x}, \mathbf{p}) \tag{22}$$

whose gradient is

$$\nabla \mathcal{L}(\mathbf{x}, \mathbf{p}, \boldsymbol{\lambda}) = \begin{bmatrix} \nabla_{\mathbf{x}} f + \nabla_{\mathbf{x}} \mathbf{c}^T \cdot \boldsymbol{\lambda} \\ \nabla_{\mathbf{p}} f + \nabla_{\mathbf{p}} \mathbf{c}^T \cdot \boldsymbol{\lambda} \\ \mathbf{c} \end{bmatrix}, \tag{23}$$

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

$$\boldsymbol{\lambda}^{\mathrm{c}=0} = - \left[\frac{\partial \mathbf{c}}{\partial \mathbf{x}}\right]^{-T} \frac{\partial f}{\partial \mathbf{x}}^T, \tag{24}$$

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 $\boldsymbol{\lambda} = \boldsymbol{\lambda}^{\mathrm{c}=0}$, we have

$$\begin{aligned} \frac{df^2}{d\mathbf{p}^2} &= \frac{d\mathbf{x}}{d\mathbf{p}}^T \frac{\partial^2 f}{\partial \mathbf{x}^2} \frac{d\mathbf{x}}{d\mathbf{p}} + \frac{\partial^2 f}{\partial \mathbf{x} \partial \mathbf{p}} \frac{d\mathbf{x}}{d\mathbf{p}} + \frac{d\mathbf{x}}{d\mathbf{p}}^T \frac{\partial^2 f}{\partial \mathbf{p} \partial \mathbf{x}} + \frac{\partial^2 f}{\partial \mathbf{p}^2} + \sum_i \frac{\partial f}{\partial \mathbf{x}_i} \frac{d^2 \mathbf{x}_i}{d\mathbf{p}^2} \\ &= \frac{d\mathbf{x}}{d\mathbf{p}}^T \nabla^2_{\mathbf{xx}} \mathcal{L} \frac{d\mathbf{x}}{d\mathbf{p}} + \nabla^2_{\mathbf{xp}} \mathcal{L} \frac{d\mathbf{x}}{d\mathbf{p}} + \frac{d\mathbf{x}}{d\mathbf{p}}^T \nabla^2_{\mathbf{px}} \mathcal{L} + \nabla^2_{\mathbf{pp}} \mathcal{L}. \end{aligned} \tag{25}$$

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

$$\sum_k \frac{\partial f}{\partial \mathbf{x}_k} \frac{d^2 \mathbf{x}_k}{d\mathbf{p}_j d\mathbf{p}_i} = \tag{26}$$

$$\begin{aligned} &= - \sum_k \frac{\partial f}{\partial \mathbf{x}_k} \left[\left(\frac{\partial \mathbf{c}}{\partial \mathbf{x}} \right)^{-1} \left(\frac{d\mathbf{x}_m}{d\mathbf{p}_j} \frac{\partial^2 \mathbf{c}}{\partial \mathbf{x}_m \partial \mathbf{x}_l} \frac{d\mathbf{x}_l}{d\mathbf{p}_i} + \frac{\partial^2 \mathbf{c}}{\partial \mathbf{x}_m \partial \mathbf{p}_i} \frac{d\mathbf{x}_m}{d\mathbf{p}_j} + \frac{d\mathbf{x}_l}{d\mathbf{p}_i} \frac{\partial^2 \mathbf{c}}{\partial \mathbf{p}_j \partial \mathbf{x}_l} + \frac{\partial^2 \mathbf{c}}{\partial \mathbf{p}_j \partial \mathbf{p}_i} \right) \right]_k \\ &= - \sum_k \left(\left(\frac{\partial \mathbf{c}}{\partial \mathbf{x}} \right)^{-T} \frac{\partial f}{\partial \mathbf{x}}^T \right)_k \left(\frac{d\mathbf{x}_m}{d\mathbf{p}_j} \frac{\partial^2 \mathbf{c}_k}{\partial \mathbf{x}_m \partial \mathbf{x}_l} \frac{d\mathbf{x}_l}{d\mathbf{p}_i} + \frac{\partial^2 \mathbf{c}_k}{\partial \mathbf{x}_m \partial \mathbf{p}_i} \frac{d\mathbf{x}_m}{d\mathbf{p}_j} + \frac{d\mathbf{x}_l}{d\mathbf{p}_i} \frac{\partial^2 \mathbf{c}_k}{\partial \mathbf{p}_j \partial \mathbf{x}_l} + \frac{\partial^2 \mathbf{c}_k}{\partial \mathbf{p}_j \partial \mathbf{p}_i} \right) \\ &= \sum_k \boldsymbol{\lambda}_k^{\mathrm{c}=0} \left(\frac{d\mathbf{x}_m}{d\mathbf{p}_j} \frac{\partial^2 \mathbf{c}_k}{d\mathbf{x}_m \partial \mathbf{x}_l} \frac{d\mathbf{x}_l}{d\mathbf{p}_i} + \frac{\partial^2 \mathbf{c}_k}{\partial \mathbf{x}_m \partial \mathbf{p}_i} \frac{d\mathbf{x}_m}{d\mathbf{p}_j} + \frac{d\mathbf{x}_l}{d\mathbf{p}_i} \frac{\partial^2 \mathbf{c}_k}{\partial \mathbf{p}_j \partial \mathbf{x}_l} + \frac{\partial^2 \mathbf{c}_k}{\partial \mathbf{p}_j \partial \mathbf{p}_i} \right). \end{aligned} \tag{27}$$

Equation (25) thus holds and, using Theorem A.1, the result follows directly. □

B BLOCK SOLVE

Using $B = 0$ and $C = 0$ in Equation (12), we have

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

Using these definitions in the third row of Equation (12), we obtain

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

and using Equation (4) with $\frac{\partial f}{\partial \mathbf{p}} = \mathbf{0}$, we finally have

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

ACKNOWLEDGEMENTS

We thank the reviewers for their valuable feedback and suggestions.

REFERENCES

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

Miklós Bergou, Basile Audoly, Etienne Vouga, Max Wardetzky, and Eitan Grinspun. 2010. Discrete viscous threads. *ACM Trans. Graph.* 29, 4, Article 116 (July 2010). https://doi.org/10.1145/1778765.1778853

Gaurav Bharaj, David I. W. Levin, James Tompkin, Yun Fei, Hanspeter Pfister, Wojciech Matusik, and Changxi Zheng. 2015. Computational design of metallophone contact sounds. *ACM Trans. Graph.* 34, 6, Article 223 (Oct. 2015). https://doi.org/10.1145/2816795.2818108

Kai-Uwe Bletzinger, Matthias Firl, Johannes Linhard, and Roland Wüchner. 2010. Optimal shapes of mechanically motivated surfaces. *Computer Methods Appl. Mechanics Eng.* 199, 5–8 (2010), 324–333.

Xiang Chen, Changxi Zheng, Weiwei Xu, and Kun Zhou. 2014. An asymptotic numerical method for inverse elastic shape design. *ACM Trans. Graph.* 33, 4, Article 95 (July 2014). https://doi.org/10.1145/2601097.2601189

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

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

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

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

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

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

Ruslan Guseinov, Eder Miguel, and Bernd Bickel. 2017. CurveUps: Shaping objects from flat plates with tension-actuated curvature. *ACM Trans. Graph.* 36, 4, Article 64 (July 2017). https://doi.org/10.1145/3072959.3073709

S.-P. Han and O. Fujiwara. 1985. An inertia theorem for symmetric matrices and its application to nonlinear programming. *Linear Algebra Appl.* 72 (1985), 47–58.

Matthias Heinkenschloss. 1996. Projected sequential quadratic programming methods. *SIAM J. Optim.* 6, 2 (1996), 373–417. https://doi.org/10.1137/0806022

Caigui Jiang, Chengcheng Tang, Hans-Peter Seidel, and Peter Wonka. 2017. Design and volume optimization of space structures. *ACM Trans. Graph.* 36, 4, Article 159 (July 2017). https://doi.org/10.1145/3072959.3073619

Martin Kilian, Aron Monszpart, and Niloy J. Mitra. 2017. String actuated curved folded surfaces. *ACM Trans. Graph.* 36, 3, Article 25 (May 2017). https://doi.org/10.1145/3015460

Shahar Z. Kovalsky, Meirav Galun, and Yaron Lipman. 2016. Accelerated quadratic proxy for geometric optimization. *ACM Trans. Graph.* 35, 4, Article 134 (July 2016). https://doi.org/10.1145/2897824.2925920

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

Tiantian Liu, Sofien Bouaziz, and Ladislav Kavan. 2017. Quasi-Newton methods for real-time simulation of hyperelastic materials. *ACM Trans. Graph.* 36, 4, Article 116a (May 2017). https://doi.org/10.1145/3072959.2990496

Mickaël Ly, Romain Casati, Florence Bertails-Descoubes, Mélina Skouras, and Laurence Boissieux. 2018. Inverse elastic shell design with contact and friction. *ACM Trans. Graph.* 37, 6, Article 201 (November 2018). https://doi.org/10.1145/3272127.3275036

Vittorio Megaro, Jonas Zehnder, Moritz Bächer, Stelian Coros, Markus Gross, and Bernhard Thomaszewski. 2017. A Computational design tool for compliant mechanisms. *ACM Trans. Graph.* 36, 4, Article 82 (July 2017). https://doi.org/10.1145/3072959.3073636

Rajaditya Mukherjee, Longhua Wu, and Huamin Wang. 2018. Interactive two-way shape design of elastic bodies. *Proc. ACM Comput. Graph. Interact. Tech.* 1, 1, Article 11 (July 2018). https://doi.org/10.1145/3203196

Przemyslaw Musialski, Christian Hafner, Florian Rist, Michael Birsak, Michael Wimmer, and Leif Kobbelt. 2016. Non-linear shape optimization using local subspace projections. *ACM Trans. Graph.* 35, 4, Article 87 (July 2016). https://doi.org/10.1145/2897824.2925886

J. Panetta, M. Konaković-Luković, F. Isvoranu, E. Bouleau, and M. Pauly. 2019. X-Shells: A new class of deployable beam structures. *ACM Trans. Graph.* 38, 4, Article 83 (July 2019). https://doi.org/10.1145/3306346.3323040

Richard Peng and Santosh Vempala. 2020. Solving Sparse Linear Systems Faster than Matrix Multiplication. (2020). arXiv:2007.10254

Yue Peng, Bailin Deng, Juyong Zhang, Fanyu Geng, Wenjie Qin, and Ligang Liu. 2018. Anderson acceleration for geometry optimization and physics simulation. *ACM Trans. Graph.* 37, 4, Article 42 (July 2018). https://doi.org/10.1145/3197517.3201290

Jesús Pérez, Miguel A. Otaduy, and Bernhard Thomaszewski. 2017. Computational design and automated fabrication of Kirchhoff-plateau surfaces. *ACM Trans. Graph.* 36, 4, Article 62 (July 2017). https://doi.org/10.1145/3072959.3073695

Nico Pietroni, Marco Tarini, Amir Vaxman, Daniele Panozzo, and Paolo Cignoni. 2017. Position-based tensegrity design. *ACM Trans. Graph.* 36, 6, Article 172 (Nov. 2017). https://doi.org/10.1145/3130800.3130809

Adriana Schulz, Harrison Wang, Eitan Grinspun, Justin Solomon, and Wojciech Matusik. 2018. Interactive exploration of design trade-offs. *ACM Trans. Graph.* 37, 4, Article 131 (July 2018). https://doi.org/10.1145/3197517.3201385

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

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

Mélina Skouras, Bernhard Thomaszewski, Peter Kaufmann, Akash Garg, Bernd Bickel, Eitan Grinspun, and Markus Gross. 2014. Designing inflatable structures. 33, 4, Article 63 (July 2014). http://dx.doi.org/10.1145/2601097.2601166

Antonio Tomas and Pascual Martí-Montrull. 2010. Optimality of Candela's concrete shells: A study of his posthumous design. *J. Int. Assoc. Shell Spatial Structures* 51 (2010), 67–77.

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

Nobuyuki Umetani, Athina Panotopoulou, Ryan Schmidt, and Emily Whiting. 2016. Printone: Interactive resonance simulation for free-form print-wind instrument design. *ACM Trans. Graph.* 35, 6, Article 184 (Nov. 2016). https://doi.org/10.1145/2980179.2980250

Henk A. Van der Vorst. 1992. Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput.* 13, 2 (1992), 631–644.

Etienne Vouga, Mathias Höbinger, Johannes Wallner, and Helmut Pottmann. 2012. Design of self-supporting surfaces. *ACM Trans. Graph.* 31, 4, Article 87 (July 2012). https://doi.org/10.1145/2185520.2185583

Huamin Wang. 2018. Rule-free sewing pattern adjustment with precision and efficiency. *ACM Trans. Graph.* 37, 4, Article 53 (July 2018). https://doi.org/10.1145/3197517.3201320

Guowei Yan, Wei Li, Ruigang Yang, and Huamin Wang. 2018. Inexact descent methods for elastic parameter optimization. *ACM Trans. Graph.* 37, 6, Article 253 (Dec. 2018). https://doi.org/10.1145/3272127.3275021

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

Jonas Zehnder, Espen Knoop, Moritz Bächer, and Bernhard Thomaszewski. 2017. Metasilicone: Design and fabrication of composite silicone with desired mechanical properties. *ACM Trans. Graph.* 36, 6, Article 240 (Nov. 2017). https://doi.org/10.1145/3130800.3130881

Yufeng Zhu, Robert Bridson, and Danny M. Kaufman. 2018. Blended cured quasi-newton for distortion optimization. *ACM Trans. Graph.* 37, 4, Article 40 (July 2018). https://doi.org/10.1145/3197517.3201359

Simon Zimmermann, Roi Poranne, James M. Bern, and Stelian Coros. 2019. Puppet-Master: Robotic animation of marionettes. *ACM Trans. Graph. (TOG)* 38, 4 (2019), 103.

Received November 2020; revised May 2021; accepted June 2021