SANM: A Symbolic Asymptotic Numerical Solver with Applications in Mesh Deformation

KAI JIA, MIT CSAIL, USA

Fig. 1. We present SANM, an open-source framework that automates and generalizes the Asymptotic Numerical Methods (ANM) to solve symbolically represented nonlinear systems via numerical continuation and higher-order approximations. We apply SANM to static equilibrium problems via continuation on gravity (1a and 1b) and controlled mesh deformation problems with an implicit homotopy formulation via continuation on control handles (1c and 1d). SANM takes a symbolic representation of the nonlinear system from the user (Listing 1) and automatically handles all the complexity of applying ANM.

Solving nonlinear systems is an important problem. Numerical continuation methods efficiently solve certain nonlinear systems. The Asymptotic Numerical Method (ANM) is a powerful continuation method that usually converges faster than Newtonian methods. ANM explores the landscape of the function by following a parameterized solution curve approximated with a high-order power series. Although ANM has successfully solved a few graphics and engineering problems, prior to our work, applying ANM to new problems required significant effort because the standard ANM assumes quadratic functions, while manually deriving the power series expansion for nonquadratic systems is a tedious and challenging task.

This paper presents a novel solver, SANM, that applies ANM to solve symbolically represented nonlinear systems. SANM solves such systems in a fully automated manner. SANM also extends ANM to support many nonquadratic operators, including intricate ones such as singular value decomposition. Furthermore, SANM generalizes ANM to support the implicit homotopy form. Moreover, SANM achieves high computing performance via optimized system design and implementation.

We deploy SANM to solve forward and inverse elastic force equilibrium problems and controlled mesh deformation problems with a few constitutive models. Our results show that SANM converges faster than Newtonian solvers, requires little programming effort for new problems, and delivers comparable or better performance than a hand-coded, specialized ANM solver. While we demonstrate on mesh deformation problems, SANM is generic and potentially applicable to many tasks.

ACM Trans. Graph., Vol. 40, No. 4, Article 79. Publication date: August 2021.
Taylor coefficient derivation as a major contribution, which is a difficult and laborious task. They have also shown that ANM converges up to orders of magnitudes faster than Newtonian solvers.

To date, however, there is no scalable tool that automates the computation of Taylor coefficients in the general case. The lack of such tools severely limits the application of ANM to new problems. In this work, we show how to solve the coefficients \( \{x_i\} \) and \( \{\lambda_i\} \) automatically and efficiently for a symbolically defined function \( f() \). More specifically, we devise techniques to establish the connection of Taylor expansion coefficients between \( x(a) \) and \( f(x(a)) \), which in fact conforms to an affine relationship for the highest-order term. We analyze a few operators important for graphics applications, including elementwise analytical functions (such as power and logarithm), matrix inverse, matrix determinant, and singular value decomposition. We speed up the system with batch computing that fits naturally into Finite Element Method (FEM) due to the same computation form shared by all quadrature points. We present a system, called SANM, as an implementation of our techniques.

We deploy SANM to solve the forward and inverse static force equilibrium problems similar to Chen et al. [2014]. In contrast to their manual derivation that only works with incompressible neo-Hookean materials, our system allows easily solving more constitutive models by changing a few lines of code, including the compressible neo-Hookean model that has a logarithm term and the As-Rigid-As-Possible energy that involves a polar decomposition. Our experimental results show that SANM achieves comparable or better performance as the hand-coded, specialized ANM solver of Chen et al. [2014]. We are unaware of efficient alternative methods for the inverse problem other than ANM. For the forward problem, an alternative is to minimize the total potential energy, and we show that SANM exhibits better performance than Newtonian energy minimizers.

We further extend the ANM framework to incorporate implicit homotopy \( H(x, \lambda) = 0 \) where \( H : \mathbb{R}^{n+1} \mapsto \mathbb{R}^n \) admits a one-dimensional solution curve. For instance, we formulate the controlled mesh deformation as an implicit homotopy problem, defined as \( H(x, \lambda) = f(x; x + \lambda \Delta x) \), where \( x \) corresponds to the initial location of control handles, \( \Delta x \) describes their user-specified movement, and \( f(\cdot) \) computes the internal elastic force. The coordinates of unconstrained nodes in the deformed equilibrium state, denoted by \( x^* \), are then governed by \( H(x^*, 1) = 0 \) and can be solved by continuation on \( \lambda \). In our experiments, SANM runs 1.41 times faster by geometric mean than Newtonian energy minimization methods.
We also demonstrate the robustness and versatility of SANM by twisting and bending a bar to extreme poses, as shown in Figure 1d. Note that ANM and Newtonian minimization methods target different problems and can not replace each other. Section 7.3 further discusses their differences.

To summarize, this paper makes the following contributions:

1. We devise analytical solutions for Taylor coefficient propagation through a few nonlinear operators on which ANM has not been applied, including singular value decomposition as a challenging case (Section 5).

2. We present a system, SANM, that automatically solves the Taylor expansion coefficients for symbolically defined functions (Section 4). SANM greatly reduces programming effort for adopting ANM-based methods (Section 6.1). SANM adopts generic and FEM-specific optimizations to improve solving efficiency further.

3. We present a novel continuation algorithm to reduce accumulated numerical error and approximation error when solving the equalional form \( f(x) + \sigma = 0 \) (Section 4.3).

4. We extend ANM to handle implicit homotopy and apply it to controlled mesh deformation problems (Section 7.2). Our experiments show that SANM often converges faster than a state-of-the-art Newtonian energy minimizer. Moreover, the numerical continuation framework of SANM directly handles constitutive models that do not support inverted tetrahedrons, which would be challenging for energy minimization methods due to undefined elastic energy at the initial guess.

SANM is available at https://github.com/jia-kai/SANM.

2 RELATED WORK

**Numerical Optimization:**
Numerical optimization has been extensively studied, and it is closely related to solving nonlinear systems. For example, we can recast solving \( f(x) = 0 \) as minimizing \( g(x) = f(x)^	op f(x) \) and apply generic minimization methods such as the Levenberg–Marquardt algorithm. On the other hand, minimizing \( f(x) \) can often be approached via solving \( \nabla f(x) = 0 \). For controlled mesh deformation problems, the internal elastic force corresponds to the gradient of the elastic potential energy with respect to node locations. Therefore, one can either directly solve a force equilibrium under Dirichlet boundary conditions (as done by SANM) or minimize the total potential energy to obtain the deformed state. We review the development of As-Rigid-As-Possible (ARAP) energy minimization as an example of improvements on numerical optimizers. Sorkine and Alexa [2007] devises a surface modeling technique minimizing the ARAP energy via alternating between fitting the rotations and optimizing the locations. Chao et al. [2010] employs a Newton trust region solver to minimize the ARAP energy. Shitengel et al. [2017] accelerates the convergence by computing a positive semidefinite Hessian via constructing a convex majorizer for a specific class of convex-concave decomposable objectives, including the 2D ARAP energy. Smith et al. [2019] presents analytical solutions for the eigensystems of isotropic distortion energies to enable easily projecting the Hessians of 2D and 3D ARAP energies to be positive semidefinite to speed up the convergence. Most optimization methods inherently build on the classic idea of using first or second order approximations and exploit problem-specific optimization opportunities. This work targets generic nonlinear solving with numerical continuation and uses higher-order approximation.

**Mesh Deformation:**
Mesh deformation control is an important and widely studied problem in graphics. For animation production that only requires plausible but not physically accurate results, the simulation performance can be improved by a variety of approaches such as model analysis [Choi and Ko 2005; Kim and James 2009], skinning [Gilles et al. 2011], and constraint projection [Bender et al. 2014; Bouaziz et al. 2014]. For physically predictive simulations, we need to stick to the formulation derived from continuum mechanics strictly, and the solver convergence rate is often improved by Hessian modification [Kim and Eberle 2020; Shitengel et al. 2017]. In this paper, we choose physically accurate elastic deformation as our target application. We approach the problem by solving a nonlinear system that encodes force equilibrium constraints.

**Numerical Continuation Methods:**
Classic numerical continuation methods include the predictor-corrector method and the piecewise-linear method. Allgower and Georg [2003] provides an introduction to this topic. The basic idea, which is to follow a solution trajectory by taking small steps, has become popular in many applications such as motion planning [Duenser et al. 2020; Yin et al. 2008], MRI reconstruction [Trzasko and Manduca 2008], and drawing assistance [Limpaecher et al. 2013]. These works typically choose a fixed step size or adopt a problem-specific step size schedule in the predictor and use classic first or second order solvers as the corrector. By contrast, asymptotic numerical methods use a higher-order approximation as the predictor without needing a corrector and adaptively choose the step size according to how well the predictor approximates the system.

**Asymptotic Numerical Methods:**
ANM has been applied to solve engineering problems in different domains, including buckling analysis [Azrar et al. 1993; Boutyour et al. 2004], vibration analysis [Azrar et al. 2002; Daya and Potier-Ferry 2001], shell and rod simulation [Lazarus et al. 2013; Zahrouni et al. 1999], and inverse deformation problems [Chen et al. 2014]. ANM assumes a quadratic system. An improvement over the standard ANM framework is to increase the range of validity of the approximation via imposing heuristics on the function behavior, such as replacing the power series with a Padé representation [Cochetin et al. 1994a; Elhage-Hussein et al. 2000; Najah et al. 1998]. When adapting ANM to new problems, one typically needs to recast their specific problems into quadratic forms by introducing auxiliary variables and deriving the expansions manually [Guillot et al. 2019]. Abichou et al. [2002] presents a review on adapting ANM for a few nonlinear functions.

Few attempts have been made to automate ANM to handle general nonlinearities. Notably, Charpentier et al. [2008] proposes an automatic differentiation framework, called Diamant, that computes the expansion coefficients by Taylor coefficient propagation via computing higher-order derivatives of the operators. Unfortunately, the Diamant approach is not readily applicable to mesh deformation problems due to the difficulty in computing higher-order derivatives of certain matrix functions such as matrix inverse or determinant used in the constitutive models. Moreover, Diamant is not designed...
with high-performance computing in mind. It only works with
calar variables, does not take advantage of the structural sparsity
in FEM problems, and is only evaluated on small-sized problems.
Lejeune et al. [2012] incorporates the Diamant approach into an
object-oriented solver to automate ANM. By contrast, SANM nat-
vously works with multidimensional variables and is accelerated
with batch computing for large-scale FEM problems. SANM also
implements a generic framework for computing the expansion coef-
ficients, which is not limited to the higher-order derivative approach
of Diamant and is capable of handling challenging matrix functions.

3 ANM BACKGROUND
This section introduces the asymptotic numerical method. We be-
gin with a toy example of a geometry problem and then formally
describe ANM. We first define the notations used in this paper in
Table 1.

Table 1. Notation definition

| Symbol | Description |
|--------|-------------|
| \( f, x \) | Scalars or scalar-valued functions |
| \( f, x \) | Vectors or vector-valued functions |
| \( f_i, x_i \) | A scalar in the vector at given index. For a function \( f(\cdot) \), we also use \( f_i \) to represent its \( i \)th Taylor coefficient, and similarly for \( f(\cdot) \) vs \( f_i \) and \( F(\cdot) \) vs \( F_i \). |
| \( f_i, X_i \) | A vector in an array of vectors |
| \( F, X \) | Matrices or matrix-valued functions |
| \( X_{ij} \) | A coefficient in the matrix at given row and column |
| \( X_{i, j} \) | The vectors corresponding to the \( i \)th row or the \( j \)th column in matrix \( X \) |
| \( \text{RMS}(x) \) | Root-mean-square of \( x \in \mathbb{R}^n \); \( \text{RMS}(x) = \sqrt{x^T x/n} \) |
| \( \text{vec}(X) \) | Flatten a matrix \( X \) into a column vector by concatenating the columns in \( X \) |
| \( ||X|| \) | Frobenius norm of the matrix \( X \), defined as \( \sqrt{\text{tr}(X^T X)} \) |
| \( \text{diag}(X) \) | A vector containing the diagonal coefficients of \( X \) |
| \( o(x) \) | The little-o notation: \( o(x) \) if \( y/x \to 0 \). |

3.1 A Circle-ellipse Intersection Problem
We illustrate ANM with a toy problem that asks for the intersection
of a circle \( e \) and an ellipse \( c \) as shown in Figure 3. The ellipse
intersects the y-axis at \( B_0 = (0, -1) \) and its radius is \( |AB| = \sqrt{8} \). To solve the problem with ANM, we first choose a continuation scheme. We start with a smaller
choice, such as the external force in static equilibrium problems.
Note that the continuation scheme is problem-specific. While there
are many choices, practical problems typically admit a “natural”
choice, such as the external force in static equilibrium problems.

Formally, our goal is to solve \((x, y)\) such that \( f_e(x, y) = f_e(x, y) = 0 \), where \( f_e(\cdot) \) and \( f_e(\cdot) \) describe the ellipse \( e \) and the circle \( c \) respectively:

\[
\begin{align*}
f_e(x, y) &= 2x^2 - 5x + y^2 - 4y - 2xy - 5 \\
f_e(x, y) &= (x + 1)^2 + y^2 - 8
\end{align*}
\]

(1)

Fig. 3. Our circle-ellipse intersection example problem. ANM uses polynomials to approximate the trace of intersection points between the ellipse and concentric circles of radius from \( \sqrt{2} \) to \( \sqrt{8} \), which is the arc \( B_0B \).

ANM introduces a variable \( \lambda \in [0, 1] \) to represent the continu-
uation. ANM traces the solution curve starting at \((x_0, y_0)\) via varying \( \lambda \) from 0 to 1 while keeping the following equations satisfied:

\[
\begin{align*}
f_e(x, y) &= 0 \lambda \\
f_e(x, y) + 6 &= 6 \lambda \\
(x_0, y_0) &= B_0 = (0, -1)
\end{align*}
\]

(2)

Geometrically, ANM continuously solves the intersection be-
tween the ellipse and a concentric circle with radius \( \sqrt{2 \cdot 6} \). ANM
parameterizes the solution curve by a variable \( a \) and approximates
\((x(\lambda), y(\lambda), \lambda(\lambda))\) with polynomial expansions at truncation order \( N \), with coefficients \( \{x_k\}, \{y_k\}, \) and \( \{\lambda_k\} \) to be solved:

\[
\begin{align*}
x(\lambda) &= 0 + \sum_{k=1}^N x_k a^k \\
y(\lambda) &= -1 + \sum_{k=1}^N y_k a^k \\
\lambda(\lambda) &= 0 + \sum_{k=1}^N \lambda_k a^k
\end{align*}
\]

(3)

We iteratively solve the coefficients by introducing the lower-
order terms in (3) into (2). We start with \((x_1, y_1, \lambda_1)\) and introduce
\( x = x_1 a, y = -1 + y_1 a, \) and \( \lambda = \lambda_1 a \) into (2):

\[
\begin{align*}
f_e(x_1 a, y_1 a - 1) &= -(3x_1 + 6y_1) a + o(a) = 0 \\
f_e(x_1 a, y_1 a - 1) + 6 &= (2x_1 - 2y_1) a + o(a) = 6x_1 a
\end{align*}
\]

(4)

We obtain two linear constraints on the three unknowns by equating
the coefficient of \( a \) in (4). Let \( u(a) = [x(a); \ y(a); \ \lambda(a)] \) denote the solution curve. ANM further identifies the path pa-
ter \( a \) as the pseudo-arclength that is the projection of the path along its tangent direction as shown in Figure 4, specifically \( a = (u(a) - u(0)) / u'(0) \), which provides the third constraint for a full-rank system:

\[
\begin{align*}
-3x_1 - 6y_1 &= 0 \\
2x_1 - 2y_1 - 6\lambda_1 &= 0 \\
x_1^2 + y_1^2 + \lambda_1^2 &= 1
\end{align*}
\]

(5)

We require \( \lambda_1 \) to be positive so that \( \lambda(a) \) is a locally increasing
function at 0, and the solution of (5) is \( x_1 = 2/\sqrt{3}, y_1 = -1/\sqrt{3}, \) and
ANM aims to solve the nonlinear system \( f(x) + \nu = 0 \) with numerical continuation, where \( f : \mathbb{R}^n \mapsto \mathbb{R}^n \) is an analytic function, and \( \nu \) is a constant. Starting from an initial solution \((x_0, \lambda_0)\) such that \( f(x_0) + \lambda_0 \nu = 0 \), continuation methods compute an approximation to trace the nearby solution curve of \( f(x) + \lambda \nu = 0 \). In the general case, the curve may not be well-conditioned under the parameterization with respect to \( \lambda \), and it is preferable to consider an arclength parameterization \( x(a) \) and \( \lambda(a) \) where \( a \) measures the arclength or pseudo-arclength [Allgower and Georg 2003].

ANM approximates \( x(a) \) and \( \lambda(a) \) by Taylor expansion at truncation order \( N \) such that \( f(x(a)) + \lambda(a) \nu \) should be sufficiently close to zero for small values of \( a \):

\[
\begin{aligned}
x(a) &= \sum_{i=0}^{N} x_i a^i \\
\lambda(a) &= \sum_{i=0}^{N} \lambda_i a^i \\
\text{s.t. } & \|f(x(a)) + \lambda(a) \nu\| = o(a^N)
\end{aligned}
\]

We require \( \lambda_1 > 0 \) so that \( \lambda(a) \) is locally increasing, and the algorithm makes progress. We then estimate the range of validity \( a_r \) such that \( x(a) \) and \( \lambda(a) \) are good approximations of the solution when \( |a| < a_r \). If \( \lambda(a_r) \geq 1 \), we can solve \( a^* \) such that \( \lambda(a^*) = 1 \) and compute the final solution \( x^* = x(a^*) \). Otherwise, when \( \lambda(a_r) < 1 \), we recompute the power series approximation starting at \( x_0 = x(a_r) \) and \( \lambda_0 = \lambda(a_r) \) and repeat the above steps.

A simple method to estimate \( a_r \), as suggested by Cochelin [1994], builds on the idea that within the range of validity, different orders of approximation should behave similarly:

\[
\frac{\|x(a)\|_{\text{order } N} - \|x(a)\|_{\text{order } N-1}}{\|x(a)\|_{\text{order } N} - x_0} < \varepsilon
\]

which leads to an approximation

\[
a_r \approx \left( \frac{\|x_0\|}{\|x(N)\|} \right)^{1/(N-1)}
\]

The equation \( \lambda(a^*) = 1 \) can be solved by a univariate polynomial root finding algorithm such as Brent’s method [Brent 2013].

The remaining part of completing the ANM algorithm is to solve the polynomials \( \{x_1\} \) and \( \{\lambda_1\} \) efficiently, which is a core contribution of this paper.

We solve the coefficients \( \{x_1\} \) and \( \{\lambda_1\} \) iteratively. Assume we are at the \( k^{\text{th}} \) iteration, where \( \{x_0, \ldots, x_{k-1}\} \) and \( \{\lambda_0, \ldots, \lambda_{k-1}\} \) have been solved. With \( x_k \in \mathbb{R}^n \) and \( \lambda_k \in \mathbb{R} \) currently unknown, we have the equation:

\[
f(x(a)) + \nu = f_k + \sum_{i=0}^{k} x_i a^i + \sum_{i=0}^{k} \lambda_i a^i = 0
\]

Assume \( f_k \) are the Taylor coefficients of \( f(x(a)) \):

\[
f(x(a)) \approx \sum_{i=0}^{k} x_i a^i + \sum_{i=0}^{k} \nu a^i + o(a^{k+1})
\]

As will be shown in Proposition 1, for an analytic function \( f(\cdot) \), there is an affine relationship between \( f_k \) and \( x_k \), specifically \( f_k = P(x_0) x_k + q(x_0, \ldots, x_{k-1}) \), where \( P(x_0) \in \mathbb{R}^{n \times n} \) is the slope matrix and \( q(\cdot) \in \mathbb{R}^n \) is the bias vector. By introducing this relationship into the original equation and requiring the coefficient of \( a^k \) to be zero, we obtain a linear system that restricts \( x_k \) and \( \lambda_k \):

\[
P(x_0) x_k + \lambda_k \nu = -q(x_0, \ldots, x_{k-1})
\]

However, the system has rank \( n \) but there are \( n+1 \) unknowns because the curve behavior with respect to its parameter \( a \) is not fully constrained. To obtain a full rank system, Cochelin [1994]
proposes to identify the path parameter \( a \) as the pseudo-arclength, similar to other numerical continuation methods [Allgower and Georg 2003]. Pseudo-arclength approximates the arclength of a curve by projecting it onto the tangent space, which constitutes the following constraint:

\[
a = (x(a) - x_0)^T x'(0) + (\lambda(a) - \lambda_0) \lambda'(0)
\]  

(12)

Introduce (6) into (12):

\[
x_0^T x_1 + \lambda_1 \lambda_1 = \| \mathbb{I} \|_{i=1}
\]  

(13)

The unknowns \( x_k \) and \( \lambda_k \) can be solved by combining (11) and (13). The solution is unique if we further require \( \lambda'(0) = \lambda_1 > 0 \).

3.3 Linearity Between Taylor Coefficients

Before proving the linearity between \( f_k \) and \( x_k \) in (10), we introduce an auxiliary definition:

**Definition 1.** Define \( \mathcal{C}_a^{(k)}[f(a)] \) to be the coefficient of \( a^k \) in the Taylor expansion of an analytic function \( f : \mathbb{R} \rightarrow \mathbb{R} \) such that:

\[
f(a) = \sum_{i \geq 0} \mathcal{C}_a^{(i)}[f(a)] a^i
\]

Then we have \( f_k = p(x_0)^T x_k + q(x_0, \ldots, x_{k-1}) \). Specifically, \( p(x_0) = \nabla f(x_0) \) and \( q(x_0, \ldots, x_{k-1}) = \mathcal{C}_a^{(k)}[f\left(\sum_{i=0}^{k} x_i a^i\right)] \).

**Proposition 1.** Let \( f : \mathbb{R}^n \rightarrow \mathbb{R} \) be an analytic function, and \( x_0, \ldots, x_{k-1} \) known real-valued \( n \)-dimensional vectors. Assume coefficients \( f_0, \ldots, f_k \) satisfy:

\[
f\left(\sum_{i=0}^{k} x_i a^i\right) = \sum_{i=0}^{k} f_i a^i + o(a^k)
\]

Then we have \( f_k = p(x_0)^T x_k + q(x_0, \ldots, x_{k-1}) \). Specifically, \( p(x_0) = \nabla f(x_0) \) and \( q(x_0, \ldots, x_{k-1}) = \mathcal{C}_a^{(k)}[f\left(\sum_{i=0}^{k} x_i a^i\right)] \).

**Proof.** We prove the univariate case for the simplicity of the notations. Our argument also applies to multivariate functions by using the multivariate Taylor theorem.

Let \( g(t) = f(x_0 + t) = \sum_{i \geq 0} g_i t^i \) where \( g_i = \frac{1}{i!} \frac{d^i g}{dt^i}[t=0] \) is the Taylor expansion coefficient and \( g_0 = f_0 = f(x_0) \). With \( t = \sum_{i=1}^{k} x_i a^i \), we rewrite:

\[
f\left(\sum_{i=1}^{k} x_i a^i\right) = g_0 + \sum_{i=1}^{k} g_i \left(\sum_{i=1}^{k} x_i a^i\right)^i
\]

To compute \( f_k \), we consider the terms that contribute to \( a^k \) in the expansion of the right side hand:

1. For terms that contain \( x_i \) and contribute to \( f_k \), it must contain \( x_i a^k \) and no other \( x_j a^l \) for which \( i > 0 \). There is only one such term, which is \( g_1 x_i a^k \). The slope \( p \) for which \( f_k = p x_k + q \) is thus \( p = g_1 = f(x_0) \).

2. The bias \( q \) consists of terms that do not contain \( x_i \), which can be computed by treating \( x_i \) as zero, or equivalently removing \( x_i a^k \) from \( t \):

\[
q = \mathcal{C}_a^{(k)}[f\left(\sum_{i=0}^{k-1} x_i a^i\right)]
\]

\[\square\]

**Remarks:** We have discussed the case of scalar functions. For a vector function \( f : \mathbb{R}^n \rightarrow \mathbb{R}^m \), the slope matrix \( F \) is its Jacobian. Note that \( p(x_0) \) only depends on the initial point \( x_0 \) and remains constant for all orders. This allows us to factorize the coefficient matrix only once to solve all the terms. The result of Proposition 1 is not new. For example, it is a direct consequence of Faà di Bruno’s formula [Roman 1980]. Here we have presented a simple proof based on elementary calculus.

3.4 Continuation with Padé Approximation

The original ANM approximates \( x(a) \) and \( \lambda(a) \) with Taylor expansions. It has been shown that replacing the Taylor expansions with Padé approximations results in a larger range of validity and thus fewer iterations.

For a scalar function \( f : \mathbb{R} \rightarrow \mathbb{R} \), its Padé approximation of order \( M + N \) approximates the function with a ratio of two polynomials, \( P_N(x) \) and \( Q_M(x) \) of degrees \( N \) and \( M \), respectively, such that \( f(x) = \frac{P_N(x)}{Q_M(x)} + o(x^{N+M}) \). The polynomials \( P_N(x) \) and \( Q_M(x) \) are determined by the first \( N + M + 1 \) Taylor coefficients of \( f \) via a set of linear constraints up to a common scaling factor. Although the Padé approximation is constructed from the information contained in the Taylor expansion, for many functions, it has a larger range of validity than the Taylor series, and it sometimes gives meaningful results even when the radius of convergence of the Taylor series is strictly zero [Basdevant 1972].

Cochelin et al. [1994a] proposes to construct a Padé approximation for the vector function \( x(a) = \sum_{i=0}^{N-1} x_i a^i \) in the form:

\[
P_N(a) = x_0 + \sum_{i=0}^{N-1} \frac{D_{N-1}(a)}{D_{N-1}(a)} x_i a^i
\]

\[
D_k(a) = \sum_{i=0}^{k} d_i a^i
\]

(14)

Najah et al. [1998] presents the process of computing the coefficients \( d_i \) from \( \{x_i\} \), which first orthonormalizes \( \{x_i\} \) and then solves \( \{d_i\} \) based on the principle that the Padé approximation should share the same lower-order Taylor coefficients. We omit the details here.

We follow the techniques presented in Elhage-Hussein et al. [2000] to determine the range of validity \( a_p \) for this Padé approximation:

\[
\frac{\|P_N(a_p) - P_{N-1}(a_p)\|}{\|P_N(a_p) - P_N(0)\|} < \epsilon
\]

(15)

The number \( a_p \) is sought via bisection in the range \( (a_r, r) \), where \( a_r \) is the range of validity of the Taylor series determined by (8), and \( r \) is the smallest positive real root of \( D_{N-1}(a) \) that can be found by numerical methods such as Bairstow algorithm [Golub and Robert-son 1967]. Note that \( P_{N-1} \) is not the first \( N - 1 \) terms in \( P_N \) but the Padé approximation computed from \( \{x_0, \ldots, x_{N-1}\} \). For all the 36 test cases presented in Section 7, Padé approximation uses 1.14 ± 0.57 fewer iterations than the original ANM formulation on average.
Algorithm 1 Taylor coefficient solver in SANM

**Input:** Computing graph $G = (V_o \cup V_e, E)$ that represents $H: \mathbb{R}^{n+1} \rightarrow \mathbb{R}^n$

**Input:** Initial value $x_0 \in \mathbb{R}^n$ and $\lambda_0 \in \mathbb{R}$ such that $H(x_0, \lambda_0) = 0$

**Input:** Truncation order $N$

**Output:** Taylor coefficients $\{\xi \}$ and $\{\lambda \}$ such that $\|H(\sum_{i=0}^N x_i a_i, \sum_{i=0}^N \lambda_i a_i)\| = o(a^N)$

Compute $P \leftarrow \frac{\partial H}{\partial x}(x_0, \lambda_0)$ by reverse mode AD on $G$

Meanwhile, record the Jacobians $p^u$ for each operator $u \in V_o$

Meanwhile, record $v \leftarrow \frac{\partial H}{\partial \lambda}(x_0, \lambda_0)$

Factorize $P$

for $k \leftarrow 1$ to $N$ do

Compute the biases $q^u_k$ for $u \in V_o$ in topological order, given $x_0, \lambda_0, \ldots, x_{k-1}, \lambda_{k-1}$ and the rules in Section 5

Compute $q_k$ by combining affine transformations ($p^u, q_k^u$) for $u \in V_o$ in topological order

Solve $x_k$ and $\lambda_k$ according to (13) and (17):

$$\begin{align*}
Px_k + \lambda_k v &= -q_k \\
\xi_k x_1 + \lambda_k \lambda_1 &= \lambda_{k+1}
\end{align*}$$

end for

### 4 THE SANM FRAMEWORK

This section presents the overall design of SANM and its two novel extensions over ANM: the handling of implicit homotopy $H(x, \lambda) = 0$ and a formulation to reduce accumulated error when solving the equation $f(x) + v = 0$. Note that this paper deals with both computing graphs and mesh networks. We use vertex to refer to a vertex in a graph and node to refer to a node in a polygon mesh.

#### 4.1 Coefficient Propagation on the Computing Graph

SANM represents a nonlinear function $f(\cdot)$ as a directed acyclic bipartite computing graph composed of predefined operators: $G = (V_o \cup V_e, E)$. The user builds the graph to specify the nonlinear function in their problem symbolically. The vertex set $V_o$ corresponds to a subset of predefined operators offered by SANM: for $u \in V_o$, there is a function $f_u$ that defines the corresponding computing. Each operator takes a fixed number of inputs, where each input can be a higher-order tensor. For example, matrix inverse takes one matrix input, and vector addition takes two vector inputs. The vertex set $V_e$ represents the variables. For a variable $v \in V_e$ and an operator $u \in V_o$, an edge $(v, u) \in E$ if the user passes $v$ as an input of the function $f_u$. An edge $(u, v) \in E$ if $v$ is an output of the function $f_u$. Many successful symbolic-numerical systems, such as Theoano [Theano Development Team 2016] and Tensorflow [Abadi et al. 2016], have adopted computing graphs to represent user-defined functions.

Besides working with the original ANM formulation $f(x) + v = 0$, SANM also supports solving the more general implicit homotopy: $H(x, \lambda) = 0$. Implicit homotopy defines a curve $x(\lambda)$. Our goal is to solve $x(1)$ given an initial value $x(0)$.

Algorithm 2 Continuation framework in SANM

**Input:** An analytic function $H: \mathbb{R}^{n+1} \rightarrow \mathbb{R}^n$

**Input:** Initial value $x_0 \in \mathbb{R}^n$ and $\lambda_0 \in \mathbb{R}$ such that $H(x_0, \lambda_0) = 0$

**Input:** Target location of the homotopy $\lambda_i > \lambda_0$

**Output:** The solution $x^* \in \mathbb{R}^n$ such that $H(x^*, \lambda_i) = 0$

$k \leftarrow 0$

while $\lambda_k < \lambda_i$ do

Compute Taylor coefficients $\{x_{ki}, \lambda_k\}$ using Algorithm 1

Compute the corresponding Padé approximation $P_N(a)$

Compute $a_r$, and $a_p$, the range of validity for these two approximations

$a_m \leftarrow \max(a_r, a_p)$

Use Taylor or Padé to approximate $f_k(a)$ and $f_k(a)$, depending on which one achieves the range $a_m$

Solve $x_k$ such that $f_k(a') = \min(f_k(a_m), \lambda_i)$

$(x_{k+1}, \lambda_{k+1}) \leftarrow (f_k(a'), f_k(a'))$

$k \leftarrow k + 1$

end while

$x^* \leftarrow x_k$

Proposition 1 implies that the $k$th order expansion of the implicit homotopy can be written as:

$$H\left(\sum_{i=0}^k x_i a_i, \sum_{i=0}^k \lambda_i a_i\right) = \sum_{i=0}^{k-1} H_i a^i + \frac{\partial H}{\partial x}(x_0, \lambda_0) \cdot x_k + \frac{\partial H}{\partial \lambda}(x_0, \lambda_0) \cdot \lambda_k + q_k a^k + \cdots$$

By requiring the coefficient of $a^k$ to be zero, we obtain the equation:

$$\frac{\partial H}{\partial x}(x_0, \lambda_0) \cdot x_k + \frac{\partial H}{\partial \lambda}(x_0, \lambda_0) \cdot \lambda_k + q_k = 0$$

Given a user-defined function $H(\cdot)$ and initial values $H(x_0, \lambda_0) = 0$, SANM computes the slope matrix $P = \frac{\partial H}{\partial x}(x_0, \lambda_0)$ via reverse mode Automatic Differentiation (AD) [Baydin et al. 2018]. It then iteratively computes the biases for each order and solves the Taylor coefficients according to (13) and (17). The bias $q_k$ is computed by merging the affine transformations of individual operators in $G$. Algorithm 1 summarizes this process.

#### 4.2 Continuation in SANM

SANM aims to solve $x^*$ such that $H(x^*, \lambda_i) = 0$ given initial values $H(x_0, \lambda_0) = 0$. We typically have $\lambda_0 = 0$ and $\lambda_i = 1$. Note that we have changed the meaning of $x_k$ and $\lambda_k$ to indicate the initial values at the $k$th iteration rather than the $k$th Taylor coefficient as in Section 4.1.

Similar to the standard ANM procedure, at the $k$th iteration, SANM uses Algorithm 1 to compute the local Taylor expansion coefficients of $x(a)$ and $\lambda(a)$ near $(x_k, \lambda_k)$ so that $H(x(a), \lambda(a)) \approx 0$. SANM then estimates the range of validity $\sigma_r$ using the standard ANM formulation in (8). SANM also computes a Padé approximation from the Taylor coefficients and estimates its range of validity as in Section 4.1.
We present algorithms to determine the affine transformations of
Algorithm 2 summarizes this process.

4.3 Reducing Error When Solving The Equational Form
When solving a nonlinear system in the original ANM formulation
\( f(x) + v = 0 \), SANM adopts a novel continuation method to reduce
accumulated numerical and approximation error. The principle is to
modify \( v \) at each iteration to incorporate the residual \( f(x_k) + \lambda_k v = 0 \). Specifically, instead of using a fixed definition \( H(x, \lambda) = f(x) + \lambda v \), we change the definition of \( H \) in Algorithm 2 at the beginning of
each iteration:

\[
H_k(x, \lambda) = f(x) + (\lambda - \lambda_k) v - (1 - (\lambda - \lambda_k)) f(x_k)
\]

(18)

Note that the loop invariant \( H_k(x_k, \lambda_k) = 0 \) is still satisfied. We use \( \lambda - \lambda_k \) to represent the progress made in the current iteration.
The final solution is found if \( \lambda = \lambda_k \). Let \( r_k := f(x_k) + v \) denote the
residual. In the continuation we solve \( H_k(x_{k+1}, \lambda_{k+1}) \approx 0 \), which implies

\[
r_{k+1} = (1 - (\lambda_{k+1} - \lambda_k)) r_k
\]

(19)

Therefore, the continuation decreases the residual. Updating \( H_k \)
at each iteration automatically accounts for numerical error and
approximation error. We also change the loop condition to \( \|r_k\| \geq \varepsilon \)
to achieve the desired solution residual \( \varepsilon \). This formulation allows us to obtain very accurate solutions such as the low residual RMS
shown in Table 4 and Table 5.

5 COMPUTING THE AFFINE TRANSFORMATIONS OF
taylor coefficients
We present algorithms to determine the affine transformations of
the highest-order Taylor coefficient for a few nonlinear operators
that are commonly used in graphics applications. As we have shown
in Section 4.1, these operators can be combined into a computing
diagram to define the nonlinearity in the target application. Although
Proposition 1 ensures the existence of the linearity being sought in
the general case, for certain operators, we can compute the affine
transformations directly without resorting to computing higher-
order derivatives. In this section, we use \( f \) to represent the nonlinear
operator under investigation and also the value of this operator with
\( x \) assumed to be the independent variable. For any variable \( x \), we use \( \{x_i\} \) to denote its Taylor coefficients with respect to the path
parameter \( a \).

Our goal is to derive an affine relationship between \( f_k \) and \( x_k \),
assuming that \( x_0, \ldots, x_{k-1} \) and \( f_0, \ldots, f_{k-1} \) are known constants.

5.1 Basic Arithmetic Operations
The affine transformations for the four basic arithmetic operations
are derived by equating the coefficient of \( a^k \) in the expansion of the equation:

- \( f = x + y \): By introducing \( f = \sum_{k=0}^{\infty} f_k a^k \) and \( y = \sum_{k=0}^{\infty} y_k a^k \), we have \( f_k = x_k + y_k \).
- \( f = x - y \): Similarly, we have \( f_k = x_k - y_k \).
- \( f = xy \): Similarly, we have \( f_k = y_0 x_k + x_0 y_k + \sum_{i=1}^{k-1} x_i y_{k-i} \).
- \( f = s/y \): We have \( x = f y \), which implies \( x_k = f_0 y_k +\)
  \( y_0 f_k + \sum_{i=1}^{k-1} f_i y_{k-i} \), and therefore \( f_k = (1/y_k) x_k - (1/y_0) y_k -\)
  \( 1/y_0 \sum_{i=1}^{k-1} f_i y_{k-i} \).

5.2 Elementwise Analytic Functions
A function \( y = f(x) \) is said to be elementwise if \( \forall i : y_i = f(x_i) \). For
such functions, we only need to derive the affine transformation for
the univariate case.

The problem of computing the Taylor coefficients of \( f(\sum_{i=0}^{k-1} x_i a^i) \)
given the coefficients \( \{x_i\} \) and the Taylor expansion of \( f(x_0 + a) \)
is known in the literature as the composition problem, for which
there exists a fast \( O((k \log k)^{3/2}) \) algorithm [Brent and Kung 1978].
However, it is not necessarily fast when \( k \) is small, and the implementa-
tion is complicated.

As shown in Griewank and Walther [2008], for most functions of
practical interest, we can find auxiliary functions \( a(x), b(x), \) and
\( c(x) \) whose Taylor coefficients \( \{a_i\}, \{b_i\}, \) and \( \{c_i\} \) can be easily
computed given \( \{x_i\} \), such that

\[
b(x)f'(x) - a(x)f(x) = c(x)
\]

(20)

The coefficients \( \{f_i\} \) are then computable in \( O(k^2) \) time via a for-
mula involving \( \{a_i\}, \{b_i\}, \{c_i\}, \) and \( \{x_i\} \). Griewank and Walther
[2008] provide a thorough treatment on this subject. We list in Ta-
ble 2 the result formulas to propagate Taylor coefficients through the
elementwise analytic functions currently used in SANM for mesh deforma-
tion applications:

| Function | Recurrence for \( f_k \) |
|----------|--------------------------|
| \( f(x) \) | \( \frac{1}{x_0} \left( x_k - \sum_{i=1}^{k-1} \frac{f_i x_k f_i}{x_i} \right) \) |
| \( \ln(x) \) | \( \frac{1}{x_0} (r f_0 x_k + \sum_{i=1}^{k-1} \frac{x_i}{x_i} (r + 1 - 1) f_i f_i x_i) \) |

Note that when \( f(x) = x^r \) and \( r \) is an integer, the recurrence is
numerically unstable when \( x \) is small. In this case, we compute the
Taylor coefficients via exponentiation by squaring for polynomials
in \( O(k^2 \log r) \) time.

5.3 Matrix Inverse
Let \( F = X^{-1} \) where \( X \in \mathbb{R}^{m \times m} \) is a square matrix. Introduce the
power series definition and rearrange the terms:

\[
\left( \sum_{i=0}^{\infty} F_i a^i \right) \left( \sum_{i=0}^{\infty} X_i a^i \right) = I
\]

(21)

The coefficient of \( a^k \) on the left hand side is \( \sum_{i=0}^{k} F_i X_{k-i} \), which
must be zero because the right hand side is a constant:

\[
\sum_{i=0}^{k} F_i X_{k-i} = 0
\]

\[
\Rightarrow F_k = -X_k^{-1} X_k X_0^{-1} = \left( \sum_{i=1}^{k} F_i X_{k-i} \right) X_0^{-1}
\]

(22)
Equation (22) explicitly defines an affine relationship between \( F_k \) and \( X_k \).

5.4 Matrix Determinant
Let \( g(a) = f(X(a)) = \det(\sum_{i=0}^{k} X_i a_i^2) \) where \( X \in \mathbb{R}^{m \times m} \). A straightforward method to compute \( g_k \) is to expand the determinant according to the Leibniz formula and compute polynomial products, which incurs exponential complexity \( O(m!k^2) \) in terms of the matrix size. Although this suffices for FEM applications in 2D or 3D (with \( m = 2 \) or \( m = 3 \)), we also present a method with polynomial complexity that is better suited for larger matrices.

The terms containing \( X_{ij} a^k \) that contribute to \( g_k \) can only be multiplied with \( X_i a_j \) where \( i \neq i \) and \( j \neq j \). The multiplier of \( X_{ij} \) is in fact \( C_{ij} \), where \( C \) is the cofactor of \( X_0 \) with \( C_{ij} \) defined as the determinant of the remaining matrix by removing the \( i \)th row and \( j \)th column of \( X_0 \). Therefore:

\[
g_k = \text{vec}(C)^\top \text{vec}(X_k) + q_k
\]  

(23)

\textbf{Computing the slope:} To efficiently compute the cofactor matrix \( C \), we use the identity of Cramer’s rule:

\[
X_0^{-1} = \frac{1}{\det(X_0)} C^\top
\]  

(24)

Computing the matrix inverse incurs numerical stability issues. Instead, we first compute the SVD decomposition \( X_0 = USV^\top \) and then compute \( C \) as:

\[
C = \det(X_0) X_0^{-\top}
= \det(U) \cdot \det(\Sigma) \cdot \det(V) \cdot U \Sigma^{-1} V^\top
= \det(U) \cdot \det(V) \cdot UD V^\top
\]  

(25)

Here \( D = \det(\Sigma) \Sigma^{-1} \) is a diagonal matrix and \( D_{il} = \prod_{j \neq l} C_{ij} \). Such a formulation avoids division of singular values and is stable even for ill-conditioned matrices.

\textbf{Computing the bias:} Similar to the argument in Proposition 1, we have \( q_k = \mathbb{E}(a)^{(k)} \left[ \det(\sum_{i=0}^{K-1} X_i a_i^2) \right] \). This is known as the polynomial matrix determinant problem. We propose an efficient solution using discrete Fourier transform, which has also been discovered by Hromčík and Šebek [1999]:

\begin{enumerate}
  \item Compute \( Y_i = \sum_{i=0}^{K-1} X_i a_i^2 \) for \( 0 \leq i < K \) with Fast Fourier Transform (FFT), where \( K \) is the next power of two after \( k \), and \( \omega_K = e^{-\frac{2\pi i}{K}} \) is a \( K \)th root of unity. This step costs \( O(k \log km) \).
  \item Use the inverse discrete Fourier transform to compute \( q_k = \frac{1}{K} \sum_{i=0}^{K-1} \sum_{i=0}^{k} a_i \omega_i^{-ik} \) in \( O(k) \) time.
\end{enumerate}

The above method computes the bias \( q_k \) in \( O((\log k + m)km) \) time.

5.5 Singular Value Decomposition
Singular Value Decomposition (SVD) generates three matrices from a single matrix input: \( U \Sigma V^\top = X \) where \( U \) and \( V \) are orthonormal matrices and \( \Sigma \) is a diagonal matrix containing the singular values in decreasing order. Here we only consider the square case \( X \in \mathbb{R}^{m \times m} \). We use \( \sigma_i = \Sigma_{ii} \) to represent the singular values.

Although the constitutive models considered in this paper do not directly use SVD, the ARAP energy, defined as \( \Psi_{\text{ARAP}}(F) \equiv \frac{1}{2} ||F - R|| \), involves a Polar Decomposition (PD) \( F = PR \). PD can be computed from SVD via \( P = U \Sigma U^\top \) and \( R = UV^\top \). SVD is also potentially useful for other applications. Therefore, we first present how to compute SVD in the SANM framework and then discuss extra modifications to compute PD more stably. The ARAP energy actually needs a rotation-variant PD to prevent reflections in \( R \) by requiring that \( \det(R) = 1 \). We also discuss how to compute such rotation-variant SVD in SANM.

An obstacle in numerical differentiation of SVD is that when there are identical singular values \( \sigma_i = \sigma_j \), the corresponding singular vectors \( U_i, U_j, V_i, \) and \( V_j \) are not uniquely determined. The Jacobians \( \frac{\partial W}{\partial x} \) and \( \frac{\partial W}{\partial y} \) in this case are thus undefined since different perturbations on \( X \) induce noncontinuous changes in \( U \) and \( V \). This case corresponds to a division by zero in the Jacobian computation, which is often circumvented by various numerical tricks [Liao et al. 2019; Papadopoulou and Lourakis 2000; Seeger et al. 2017].

In graphics applications, however, identical singular values occur frequently. For example, the singular values of the deformation gradient matrix in isotropic stretching are all identical. As a remedy, we propose to use an alternative form of SVD that includes \( U \Sigma V^\top \) directly, which we denote by SVD-W:

\[
U \Sigma U^\top W = X
W = UV^\top
\]

(26)

Note that \( W \) is also the rotation matrix in the polar decomposition of \( X \), which is unique when \( X \) is invertible and the Jacobian \( \frac{\partial W}{\partial X} \) is thus well-defined.

Now we present the derivation of affine transformations from \( X_k \) to \( U_k \Sigma_k \) and \( W_k \). We are not going to give a final equation because it will be too complex and repeat most of the derivation. Instead, we focus on explaining the overall procedure for deriving these affine transformations.

We start by expanding the product \( U \Sigma U^\top W \) and extracting the coefficient of \( a^k \), which should be equal to \( X_k \):

\[
X_k = \sum_{\substack{a+b+c+d+k \leq k \\text{min}(a,b,c,d) \geq 1}} U_a \Sigma_b U_c^\top W_d + E
\]

(27)

\[
E = U_k \Sigma_0 U_0^\top W_0 + U_0 \Sigma_k U_0^\top W_0
+ U_0 \Sigma_d U_0^\top W_k
\]

(28)

We define \( F = U_0^\top E W_0^\top U_0 \). Introduce (28) to the right hand side:

\[
F = U_k^\top U_k \Sigma_0 + \Sigma_k + \Sigma_0 U_k^\top U_k W_k W_0^\top U_0
\]

(29)

Note that (27) establishes an affine relationship between \( X_k \) and \( E \), and \( F = U_0^\top E W_0^\top U_0 \) is also linear. We now only need to seek the affine transformations from \( F \) to \( U_k \Sigma_k \) and \( W_k \).
Expand the orthogonality constraints $U^* U = I$ and $W^* W = I$:

\[
U_0^T U_k + U_k^T U_0 + B_u = 0 \quad (30)
\]
\[
W_0^T W_k + W_k^T W_0 + B_w = 0 \quad (31)
\]

where $B_u = \sum_{i=1}^{k-1} U_i^T U_{k-i}$ and $B_w = \sum_{i=1}^{k-1} W_i^T W_{k-i}$

**Solving $\Sigma_k$:** From the constraints (30) and (31), we have

\[
\text{diag}(U_0^T U_k) = -\frac{1}{2} \text{diag}(B_u) \quad (32)
\]
\[
\text{diag}(U_0^T W_k W_0^T U_0) = -\frac{1}{2} \text{diag}(U_0^T B_w U_0) \quad (33)
\]

Introducing (32) and (33) into (29) allows us to solve $\text{diag}(\Sigma_k)$ from $d_f(\Sigma)$.

**Solving $W_k$:** Because $U_0^T U_k \Sigma_0 + \Sigma_0 U_0^T U_k$ is symmetric, we can cancel this term in (29) by subtracting $F^\top$ from $F$:

\[
F - F^\top = \Sigma_0 U_0^T W_k W_0^T U_0 - U_0^T W_0 W_k^T U_0 \Sigma_0 \quad (34)
\]

From (31) we have $W_k^T = -B_w W_0^T W_k W_0^T U_0 + U_0^T W_k W_0^T U_0$. Introduce it to (34):

\[
F - F^\top - U_0^T W_0 B_w W_0^T U_0 = \Sigma_0 U_0^T W_k W_0^T U_0 + U_0^T W_k W_0^T U_0 \Sigma_0 \quad (35)
\]

Note that (35) is a Sylvester equation in the form $\Sigma_0 M + M \Sigma_0 = A$ with $M = U_0^T W_k W_0^T U_0$. The solution is $M_{ij} = \frac{A_{ij}}{\sigma_i + \sigma_j}$ and $W_k = U_0 M_{ij} W_0$.

**Solving $U_k$:** Introducing $U_k^T = -B_u U_0^T U_k + U_k^T U_0$ (derived from (30)) and the solutions of $W_k$ and $\Sigma_k$ into (29) results in another Sylvester equation: $\Sigma_0 U_0^T U_k - U_0^T U_k \Sigma_0 = B$, with the solution

\[
(U_0^T U_k)_{ij} = \frac{B_{ij}}{\sigma_i - \sigma_j}. \quad (36)
\]

5.5.2 Implementation Notes. Although the SVD-W formulation provides a numerically stable expression to compute the Jacobian $\frac{\partial W}{\partial X}$, the biases $W_k$ still depend on the numerically unstable $U_0$ via (27). The polar decomposition formulation does not suffer from this problem because $P = U_0 U^*$ is unique in the presence of equal singular values as long as $X$ is invertible. In the implementation, we transparently switch to the polar decomposition formulation to compute $W_k$ in the SVD-W operator when the outputs $U$ and $\Sigma$ are not needed by other operators in the computing graph. We adopt the Lorentzian broadening [Liao et al. 2019] $x/y \rightarrow x/y/(y^2+\epsilon)$ with $\epsilon = 10^{-12}$ when computing the divisions in solving the Sylvester equations.

The rotation-variant SVD requires $\det(W) = 1$ so that $W$ is a proper rotation matrix. It is traditionally obtained by negating the last singular value and the corresponding left-singular or right-singular vector if $\det(W) = -1$ [Kim and Eberle 2020]. However, when the last singular value is identical to another singular value, the Jacobian $\frac{\partial W}{\partial X}$ becomes undefined because there are multiple singular vectors for this singular value, and it is arbitrary to negate one of them. From a numerical perspective, we need to compute $1/(\sigma_i + \sigma_j)$ in the Jacobian with some $\sigma_i + \sigma_j = 0$. To improve numerical stability, we modify the rotation-variant SVD computation by broadening identical singular values together and preferring to negate all singular values and vectors in a group of an odd size. In the 3D case, there is an odd number of singular values, and therefore an odd-sized group must exist.

6 THE SANM SYSTEM

We use C++ to implement SANM. This section discusses a few design choices that support efficient mesh deformation applications in SANM.

6.1 API Design

SANM adopts a define-and-run paradigm. The user describes a nonlinear system symbolically and provides initial values and input/output transformations. SANM automatically solves the system using the extended ANM framework described in previous sections.

One of the most important public APIs of SANM is for building the symbolic computing graph to represent the nonlinear system of interest. We adopt an object-oriented design to enable intuitive and efficient computing graph building. We use objects in the program to represent variable vertices in the computing graph. We also provide overloading for common arithmetic operators. Listing 1 shows a SANM code excerpt of expressing the first Piola-Kirchhoff tensors for a few constitutive models. SANM only requires the user to provide the C++ object that represents the whole nonlinear system, the sparse affine transformations on the inputs and outputs (see Section 6.2), and the initial values. The user does not need to modify the solver to work on different tasks.

SANM significantly reduces programming effort for applying ANM. We roughly measure programming effort by the number of lines of C++ code. With SANM, the whole FEM solver for all elastic deformation applications in this paper, including auxiliary functionalities such as mesh input/output and tetrahedron processing, needs only 1.5K lines of code without using external geometry manipulation libraries. The SANM library itself has about 7.5K lines of
The computing graph then contains a single operator that computes matrix multiplication for all elements together:

\[
\mathbf{F} = \text{batched_matmul}(\mathbf{D}_m, \mathbf{D}_m^{-1}).
\]

In our mesh deformation applications, the unknown vector \( \mathbf{x} \) represents node coordinates. We use a sparse affine transformation \( \mathbf{A} \) to map the coordinates to shape matrices and another transformation \( \mathbf{B} \) to map from stress tensors to nodal force: \( f(\mathbf{x}) = \mathbf{B}(\mathbf{F}(\mathbf{A}(\mathbf{x}))) \) where \( \mathbf{F} \) represents the Piola–Kirchhoff stress tensors, \( \mathbf{F} \) represents the deformation gradients, and \( \mathbf{P}(\mathbf{F}(\cdot)) \) is computed in a batched manner. The sparse affine transformations \( \mathbf{A} \) and \( \mathbf{B} \) are provided as input/output transformations to the SANM solver.

Batch computing improves performance by better utilizing the hardware, although it does not reduce computational complexity. Most modern CPUs support Single Instruction Multiple Data (SIMD) parallelism, and GPUs are designed to process large amounts of data in parallel. Without batch computing, such hardware capability can hardly be utilized by the small matrices occurring in finite element analysis. Batch computing also amortizes the overhead of computing graph traversing in Algorithm 1.

SANM supports parallel computing by splitting data on the batch dimension, which is managed by the framework and is oblivious to individual operator implementations. Figure 5 compares solving times achieved with different numbers of threads, which shows that SANM exhibits modest scalability on practical workloads.

### 6.3 Performance Optimizations

We design SANM with high-performance computing in mind. Here we discuss other optimizations besides batch computing.

**Efficient numerical computing primitives:** SANM provides an abstraction of numerical computing primitives. The implementations of computing graph operators invoke these primitives instead of directly working on numerical data. This paradigm allows us to separate numerical algorithm description from performance engineering. Currently, we use Eigen [Guennebaud et al. 2010] and Intel Math Kernel Library to implement the computing primitives on CPU with Single Instruction Multiple Data (SIMD) optimizations. We can easily extend SANM to support GPU by adding another GPU backend for the primitives without modifying implementations of
operators or the ANM solver. This abstraction also allows SANM to benefit from other research on optimizing tensor computing performance, such as recent related research in deep learning [Chen et al. 2018; Jouppi et al. 2017].

**Automatic memory management with copy-on-write**: SANM automatically manages tensor memory by reference counting, with eager memory sharing and copy-on-write to simplify programming without sacrificing performance. When a tensor object is copied, only a new reference is stored in the destination. When a tensor object is modified, SANM makes a private copy before modification if the reference count is greater than one.

**Sparse affine transformations**: We exploit the structural sparsity when computing the Jacobians and the affine transformations in Algorithm 1. For a batch-packed tensor with dimensions $n \times m \times m$, we use an $n \times m^2 \times m^2$ tensor to represent its Jacobian instead of using a full $nm^2 \times nm^2$ matrix because each matrix in the batch is independent of each other. Our sparse Jacobian representation significantly reduces memory usage when $n$ is large. It is also friendly to batch computing. Furthermore, we use a compact $n \times m^2$ matrix to represent Jacobians for elementwise operators.

**Special handling of zero tensors**: Zero tensors frequently occur, such as being used as the initial accumulation value. SANM retains a special buffer shared by all zero-initialized tensors (also with reference counting and copy-on-write). Thus checking whether a tensor is all zero can be easily done by comparing the buffer address. This design allows implementing a fast path for handling zero inputs in elementary arithmetic operators, such as $x + 0 = x$ and $x \cdot 0 = 0$. This optimization leads to a 7.91% speedup in our experiment.

### 6.4 Future Performance Optimizations

We discuss other optimizations that are not yet implemented but likely to be helpful. Thanks to the define-and-run paradigm, SANM users can benefit from future optimizations by simply updating their SANM library without modifying their application code.

**Computing graph optimization**: Since the user symbolically defines the computing graph, SANM can optimize the graph before starting numerical computation. For example, we can simplify arithmetic expressions. We can also fuse arithmetic operators with just-in-time compilation. There is a large body of research on traditional compiler optimization [Lattner and Adve 2004] and recent tensor compiler optimization in deep learning [Lattner et al. 2020] that may benefit future SANM optimizations.

**Reducing memory usage**: Currently, we store all the intermediate Taylor coefficients in memory, which incurs some memory overhead. A possible improvement is setting up checkpoints on the computing graph and recomputing the Taylor coefficients between checkpoints each time. A good choice of checkpoints might induce little computational cost ($O(1)$ times the original cost) while saving lots of memory ($O(1/\sqrt{N})$ relative memory usage) for a computing graph with a chain of length $N$ [Chen et al. 2016].

### 7 MESH DEFORMATION APPLICATIONS

We evaluate SANM on a few volumetric mesh deformation problems. We first briefly review the basics of elastic deformation analysis. Readers may refer to Bonet and Wood [2008]; Kim and Eberle [2020]; Sifakis and Barbic [2012] for a more thorough treatment on this subject.

We consider 3D deformation of hyperelastic materials, for which the work done by the stresses during a deformation process only depends on the initial and final state. A constitutive model relates the elastic potential energy density $\Psi$ to the deformation gradient $F$.

$$\Psi = \frac{1}{2} \tr (\mathbf{D} \mathbf{F}^T)$$

where $\mathbf{D}$ is the deformed shape matrix (computed from the deformed tetrahedron) and $\mathbf{F}$ is the reference shape matrix (computed from the rest tetrahedron). A shape matrix of a tetrahedron packs the three column vectors from one vertex to the other three. The elastic force $f_{ij} \in \mathbb{R}^3$ exerted by a single tetrahedron $i$ to its $j$th node is then derived by taking the gradient of the potential energy with respect to the node coordinates: $f_{ij} = \frac{\partial \Psi(f_i)}{\partial x_j}$. We can obtain the formulation for the internal elastic force at a node $i$ by combining forces exerted by neighboring tetrahedrons, which equals the following:

$$f_{i} = \sum_{t \in N_i} P(F_t) n_{t,i}$$

where $N_i$ is the set of adjacent tetrahedrons containing node $x_i$, $P(F) := \frac{\partial \Psi(F)}{\partial F}$ is the first Piola–Kirchhoff stress tensor of the constitutive model, and $n_{t,i}$ is the outward area-weighted normal vector at node $x_i$ of the tetrahedron $t$ in the undeformed state.

To solve the inverse deformation problem that seeks a rest shape which deforms to a given shape, we introduce the Cauchy stress tensor $\sigma$ that linearly relates the elastic force to the deformed state:

$$f_{i} = \sum_{t \in N_i} \sigma(F_t) n_{t,i}$$

$$\sigma(F) = \frac{1}{\det(F)} P(F) F^T$$

where $n_{t,i}$ is the outward area-weighted normal vector at node $x_i$ of the tetrahedron $t$ in the given deformed state.
Table 3. Comparing with the hand-coded, specialized ANM solver of Chen et al. [2014] on gravity equilibrium problems. In this table, inv. and fwd. mean inverse and forward problems respectively, and mt4 or mt6 indicate using 4 or 6 threads for parallel computing. Since the code of Chen et al. [2014] no longer compiles on modern systems, we directly use the data reported in their paper. We use the models included in their open-source release and exclude models with additional external force because their force description file is in a custom binary format. We ran SANM on a server with two Intel Xeon Platinum 8260Y CPUs (2.5GHz - 3.8GHz), while Chen et al. [2014] used a desktop PC with an Intel i7-3770K CPU (3.5GHz - 3.9GHz). We use the same truncation order $N = 20$ as Chen et al. [2014]. The data show that SANM delivers comparable, if not better, performance as a hand-coded and manually optimized ANM solver.

| Model   | Ours: SANM | Chen et al. [2014] |
|---------|------------|-------------------|
|         | #Iter. | Time | Time (mt4) | #Iter. | Time (mt6) |
| inv. bar    | 2      | 2.95 | 1.03       | 2      | 2.38       |
| inv. plant  | 2      | 7.36 | 2.70       | 3      | 7.07       |
| fwd. bar    | 2      | 2.90 | 0.98       | 3      | 3.25       |
| fwd. plant  | 2      | 7.15 | 2.69       | 4      | 9.27       |

This paper considers three constitutive models: the compressible neo-Hookean energy (denoted by NC), the incompressible neo-Hookean energy (denoted by NI), and the As-Rigid-As-Possible energy (denoted by ARAP). We list their first Piola–Kirchhoff stress tensors:

$$P_{NC}(F) = \mu (F – F^\top) + \lambda \log(J) F^\top$$  \hspace{1cm} (43)

$$P_{NI}(F) = \mu F^{-\frac{2}{3}} \left( F – \frac{1}{3} ||F|| F^\top \right) + \kappa J(F – 1) F^\top$$  \hspace{1cm} (44)

$$P_{ARAP}(F) = \mu (F – R)$$  \hspace{1cm} (45)

$$J := \det(F)$$

Note that ARAP (45) needs a rotation-variant polar decomposition $F = RS$ such that $R$ is a proper rotation matrix with $\det(R) = 1$. The parameters, $\lambda$ (Lamé’s first parameter), $\mu$ (Lamé’s second parameter), and $\kappa$ (the bulk modulus) are all determined by the physical properties of the material.

7.1 Forward and Inverse Static Equilibrium Problems

Given a static external force, we consider the problems of solving the rest shape given the deformed shape (the inverse problem) and solving the deformed shape given the rest shape (the forward problem) similar to Chen et al. [2014]. Formally, let $\bar{x}$ denote the rest shape, $x$ the deformed shape, $f(\bar{x}, x)$ the internal elastic force, and $f_{ext}$ the external force. We solve $f(\bar{x}; x_b), \{x; x_b\} + f_{ext} = 0$ given either $\bar{x}$ or $x$, where $x_b$ contains fixed boundary nodes. This problem naturally fits into the numerical continuation framework by replacing $f_{ext}$ with $\lambda f_{ext}$. We consider static equilibrium under gravity and set $f_{ext}$ as the per-node gravity.

Chen et al. [2014] has shown that ANM is tens to thousands of times faster than the Levenberg-Marquardt algorithm on the inverse problem, and it is also roughly ten times faster than an implicit Newmark simulation with kinetic damping [Umetani et al. 2011] on the forward problem. Table 3 compares SANM with the hand-coded and manually optimized ANM solver of Chen et al. [2014], which shows that SANM achieves comparable or better performance while automatically solves the problem. We also evaluate SANM on a large Armadillo model with 221,414 nodes and 696,975 tetrahedrons, which is nearly ten times larger than the models used in Chen et al. [2014]. Figure 5 presents the solving time of the forward problem using different numbers of threads. Figure 2 shows the intermediate states for a few values of $\lambda$ in the continuation.

7.1.1 Comparison with Newton’s methods. We compare with more methods on the forward problem. The minimum total potential energy principle dictates that the equilibrium state is the minimizer of the total potential energy, including the elastic potential energy and the gravitational potential energy. Therefore, an alternative method for the forward problem is to solve $\arg \min_{\bar{x}} (Ψ(\bar{x}) – g^\top x)$ where $g$ is the per-node gravity. We implement Newton’s method with backtracking line search to solve the minimization. We also evaluate positive-semidefinite Hessian projection with a state-of-the-art derivation of per-tetrahedron analytic eigensystems for the elastic energy functions [Smith et al. 2019]. Note that the energy minimization method does not apply to the inverse problem due to the lack of corresponding global energy. We also compare with directly minimizing $\|f(x) + g\|$ by the Levenberg-Marquardt algorithm.

We run the experiments on a desktop PC with an AMD Ryzen Threadripper 2970WX CPU. All the implementations are compiled with the same compiler, use the same linear algebra libraries, and use a single thread. Newton’s method uses an LU solver, and the projective Newton uses a faster LLT solver due to the guaranteed positive definiteness of projected Hessians. We use techniques described in Section 4.3 to reduce the error of SANM. We set the truncation order $N = 20$. We set the convergence threshold to be $\epsilon = 10^{-10}$ for the RMS of the force residual. We find that energy minimization with Newtonian methods often fails to converge to such a small force residual due to vanishing step sizes near the optimum, and therefore we stop them if either the RMS of the force residual or the change of $x$ in one iteration drops below $10^{-6}$. We then use additional Gauss-Newton iterations to fine-tune the solution. Table 4 presents the comparisons, which shows that SANM converges faster than the considered alternative methods in most cases and achieves low residual without additional refinement.

7.2 Controlled Mesh Deformation

We demonstrate implicit homotopy solving on controlled mesh deformation problems. The problem asks for the equilibrium state when specific nodes are moved to given locations. The constrained nodes, called the control handles, are usually specified by a user so that the elastic body can be deformed into the desired pose. This problem is typically solved with an energy minimization framework that minimizes the total elastic potential energy.

We propose an alternative approach with implicit homotopy. Given the rest shape of the body as $x_b$, the initial position of control handles as $x_c$, and the target position of control handles as $x_t$, we
Table 4. Performance comparison on forward gravity equilibrium problems. We limit the Levenberg-Marquardt algorithm to use no more than 1000 iterations. The NC and NI materials refer to compressible and incompressible neo-Hookean materials respectively. RMS(\text{f}) refers to the root-mean-square value of force residuals on unconstrained nodes. The bracketed numbers indicate the Gauss-Newton refinement iterations (limited to be 20). Proj. Newton uses per-tetrahedron Hessian projection derived from Smith et al. [2019]. SANM achieves the low residual without extra refinement thanks to the techniques presented in Section 4.3. Bold times mark the fastest methods and italic times mark the second fastest methods. SANM achieves an average speedup of 2.21 by geometric mean.

| Mesh     | Material | Ours: SANM #Iter Time (s) | RMS(\text{f}) | Newton #Iter Time (s) | RMS(\text{f}) | Proj. Newton #Iter Time (s) | RMS(\text{f}) | Levenberg-Marquardt #Iter Time (s) | RMS(\text{f}) | SANM Speedup |
|----------|----------|---------------------------|---------------|-----------------------|---------------|-----------------------------|---------------|-----------------------------------|---------------|--------------|
| armadillo-s  | ARAP  | 4 | 10.07 | 2.7e-13 | 27(1) | 24.92 | 1.5e-14 | 73(2) | 52.35 | 1.7e-14 | 1000 | 3785.59 | 4.9e-6 | 2.47 |
| bar       | NC     | 2 | 6.55  | 5.0e-12 | 21(1) | 17.60 | 4.3e-14 | 36(2) | 25.94 | 5.4e-14 | 1000 | 3706.22 | 3.2e-6 | 2.69 |
| bifur3    | ARAP   | 1 | 2.91  | 6.8e-15 | 32(1) | 11.41 | 4.7e-13 | 44(2) | 12.05 | 3.0e-12 | 1000 | 3415.21 | 3.2e-6 | 2.46 |
| bifur3    | NC     | 2 | 2.72  | 6.7e-15 | 33(1) | 11.39 | 2.6e-14 | 45(2) | 12.11 | 3.9e-12 | 1000 | 3355.08 | 3.5e-6 | 2.32 |
| bob       | NC     | 1 | 1.89  | 1.0e-11 | 5(1)  | 2.82  | 1.0e-11 | 7(1)  | 3.14  | 3.4e-11 | 336 | 563.69  | 1.1e-11 | 1.49 |
| human     | NC     | 2 | 2.11  | 9.9e-12 | 5(1)  | 2.85  | 1.0e-11 | 7(1)  | 3.10  | 3.3e-11 | 336 | 571.27  | 1.1e-11 | 1.35 |
| plant     | NC     | 2 | 4.10  | 6.8e-15 | 10(1) | 7.35  | 6.9e-15 | 17(1) | 9.22  | 5.0e-12 | 207 | 563.91  | 3.4e-11 | 1.79 |
|          | ARAP   | 8 | 41.47 | 4.1e-12 | 9(6)  | 17.55 | 7.7e-13 | 91(2) | 129.33 | 1.3e-14 | 1000 | 6695.33 | 7.0e-6 | 0.42 |
|          | NC     | 2 | 13.73 | 1.1e-13 | 11(1) | 6.92  | 3.9e-14 | 14(1) | 7.05  | 1.8e-11 | 275 | 672.32  | 1.1e-12 | 3.21 |
|          | ARAP   | 6 | 15.25 | 4.8e-15 | 60(2) | 57.14 | 2.0e-12 | 88(3) | 62.59 | 1.0e-10 | 1000 | 2846.89 | 3.5e-5 | 3.75 |
|          | NC     | 2 | 6.66  | 9.8e-14 | 49(1) | 44.26 | 3.3e-14 | 53(3) | 39.23 | 3.4e-11 | 1000 | 2586.42 | 4.3e-6 | 0.82 |

The solution contains inverted tetrahedrons.

Table 5. Performance comparison on controlled mesh deformation problems. This table uses similar notations as Table 4. Note that Newton’s minimization methods do not work with neo-Hookean energies because the initial guess contains inverted tetrahedrons. SANM uses equation solving presented in Section 4.3 to refine the solution, while Newton’s methods use Gauss-Newton iterations for refinement. SANM achieves an average speedup of 1.41 by geometric mean.

| Mesh     | Ours: SANM (ARAP) #Iter Time (s) | RMS(\text{f}) | Newton (ARAP) #Iter Time (s) | RMS(\text{f}) | Proj. Newton (ARAP) #Iter Time (s) | RMS(\text{f}) | SANM Speedup |
|----------|----------------------------------|---------------|---------------------------------|---------------|-----------------------------------|---------------|--------------|
| armadillo-s  | 3(1)  | 6.18   | 7.0e-15 | 13(1) | 12.12 | 1.3e-12 | 10(1) | 7.73 | 2.8e-13 | 1.25 | 3(1) | 6.98 | 1.5e-15 | 4(1) | 10.72 | 6.6e-14 |
| bar       | 8(1)  | 5.89   | 4.5e-12 | 23(20) | 7.02  | 1.3e+06 | 57(2) | 12.13 | 4.5e-12 | 2.06 | 7(1) | 5.34 | 3.6e-11 | 7(1) | 6.24 | 3.4e-11 |
| bifur3    | 6(1)  | 8.56   | 1.3e-12 | 43(20) | 21.76 | 5.3e+03 | 27(2) | 10.39 | 1.3e-12 | 1.21 | 6(1) | 9.15 | 1.1e-11 | 6(1) | 10.45 | 1.1e-11 |
| bob       | 3(1)  | 4.71   | 1.4e-11 | 17(11) | 10.16 | 5.6e-15 | 13(2) | 6.97  | 5.4e-15 | 1.48 | 4(1) | 6.78 | 1.1e-12 | 5(1) | 9.54 | 1.4e-12 |
| human     | 3(1)  | 11.42  | 1.9e-15 | 9(1)  | 14.45 | 3.2e-12 | 8(1)  | 10.87 | 6.1e-11 | 0.95 | 4(1) | 22.17 | 1.3e-14 | 4(1) | 22.79 | 1.4e-14 |
| plant     | 6(2)  | 13.59  | 5.7e-15 | 28(1) | 24.39 | 9.1e-13 | 83(3) | 60.19 | 6.8e-13 | 1.79 | 5(1) | 15.04 | 1.0e-12 | 7(2) | 21.66 | 3.7e-14 |

The solution contains inverted tetrahedrons.

The solution contains inverted tetrahedrons.

\[ H(x, \lambda) = f\left( \frac{x_0}{x_c}, \lambda x + \lambda (x_f - x_c) \right) = 0 \]  

where \( f(x, x) \) computes the internal elastic force given the rest shape \( x \) and the deformed shape \( x \). Numerical continuation of \( \lambda \) from 0 to 1 solves the coordinates of unconstrained nodes, and each intermediate configuration is a valid equilibrium state with the control handles on a linear path from \( x_c \) to \( x_f \). Computing an intermediate state at \( \lambda_i \) only requires solving an equation \( \lambda(a) = \lambda_i \) and then evaluate \( x(a) \), which incurs negligible extra cost. After solving the homotopy, we refine the solution by solving a static equilibrium problem with zero external force using the improved equation solving presented in Section 4.3 (with truncation order \( N = 6 \)) to reduce the force residual.

We compare SANM with Newton’s methods on a few 3D models with manually specified target positions of control handles, such as the deformed Bob shown in Figure 1c. Experimental settings are similar to those in Section 7.1.1. Newton’s methods directly minimize \( \Phi(x; x_f) \) starting at \( \Phi(x_0; x_f) \). We do not evaluate the Levenberg-Marquardt algorithm since it is too inefficient compared
to others. Note that the initial guess $x = x_0$ for Newton’s methods induces inverted tetrahedrons. Unfortunately, the neo-Hookean energies do not handle this case (corresponding to $J < 0$ in (43) and (44)), and therefore we do not evaluate Newton’s methods on them. Furthermore, a straightforward Newton’s method without Hessian projection sometimes fails on the ARAP energy because the inverted tetrahedrons cause indefinite Hessian with more negative eigenvalues that obstruct the optimization progress. By contrast, numerical continuation in SANM ensures a smooth process where no tetrahedron gets inverted, and therefore it also works for neo-Hookean energies. Table 5 presents the results, which shows that SANM is more robust and more efficient in most cases.

We demonstrate the robustness of SANM on a problem with larger deformation. We twist a horizontal bar by 360 degrees and then bend it as shown in Figure 6. Note that the target boundary configuration is indistinguishable from another one which has only the bending but no twisting. SANM naturally handles this case by using a piecewise-linear description of the movement path. However, to apply energy minimization methods, a similar but arguably less principled continuation scheme (such as minimizing the energy in multiple stages) or more complicated initialization strategies are needed to resolve the rotation ambiguity.

Fig. 6. Twisting a horizontal bar by 360 degrees and then bending it, with three constitutive models from left to right: compressible neo-Hookean, incompressible neo-Hookean, and ARAP.

7.3 Discussion
Our experiments show that SANM delivers shorter solving times than Newton’s methods. On the 24 comparison experiments (including both forward equilibrium problems and controlled deformation problems), SANM achieves an average speedup of 1.97 by geometric mean compared to the fastest alternative method for each case. On average, SANM spends 23.43% ± 1.56% of its running time in the sparse linear solver, while most of the other time is used by Taylor coefficient computation that can be further improved.

SANM and Newton’s methods target different problems, and they can not completely replace each other. SANM solves nonlinear systems via numerical continuation, while Newton’s methods typically solve minimization problems. Moreover, numerical continuation allows easily computing intermediate equilibrium states almost for free. By comparison, the intermediate states of a Newtonian solver are less interpretable, but such solvers might reduce the energy in early iterations and thus quickly produce visually plausible results.

Energy minimization does not apply to all of our experiment problems. We are unaware of any global energy suitable for the inverse static equilibrium problem. For controlled deformation problems, finding a proper initial guess for energy minimization becomes non-trivial for certain energies that can not handle inverted elements. We also need to take special care when applying energy minimization to target configurations that involve ambiguity, such as rotations. By contrast, SANM directly and efficiently handles these cases with numerical continuation.

8 CONCLUSION
The asymptotic numerical method is a powerful numerical continuation method for solving nonlinear systems. Prior to our work, a major obstacle of applying ANM was the difficulty in deriving the Taylor coefficients. We have shown that this process can be fully automated and generalized to handle a large family of nonlinearities. We also extend the ANM formulation to handle implicit homotopy. Moreover, we implement an efficient and automatic ANM solver, SANM, that delivers comparable or better performance than a hand-coded, manually optimized, and specialized ANM solver. Although energy minimization targets different problems from SANM in general, we also compare SANM with energy minimization via Newton’s methods on a few problems and show that SANM performs favorably.

With our tool, one can explore ANM on many applications in various fields with little effort. SANM can contribute to improvements in numerical solving in many systems. It may also inspire deeper theoretical understanding and further improvement of ANM.

ACKNOWLEDGMENTS
We thank all the anonymous reviewers for providing the detailed review feedback, which has greatly improved the quality of this work. We would also like to thank Changxi Zheng, Wojciech Matusik, Liang Shi, and Martin Rinard for the helpful technical discussions and Lingxiao Li for proofreading. This work was funded by the project “Automatically Learning the Behavior of Computational Agents” (MIT CO 6940111, sponsored by Boeing with sponsor ID #Z0918-5060).

REFERENCES
Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. TensorFlow: A system for large-scale machine learning. In 12th USENIX symposium on operating systems design and implementation (OSDI 16). 265–283.
H Abouh, H Zahrouni, and M Potier-Ferry. 2002. Asymptotic numerical method for problems coupling several nonlinearities. Computer Methods in Applied Mechanics and Engineering 191, 51-52 (2002), 5795–5810.
Eugene I. Allgower and Kurt Georg. 2003. Introduction to numerical continuation methods. SIAM.
L Armar, EH Bouyoun, and M Potier-Ferry. 2002. Non-linear forced vibrations of plates by an asymptotic–numerical method. Journal of Sound and Vibration 252, 4 (2002), 657–674.
L. Armar, B Cochehill, N Dumil, and M Potier-Ferry. 1993. An asymptotic-numerical method to compute the postbuckling behaviour of elastic plates and shells. International Journal for Numerical Methods in Engineering 36, 8 (1993), 1251–1277.
JL Basdevant. 1972. The Padé approximation and its physical applications. Fortschritte der Physik 20, 5 (1972), 283–331.
Atilim Gunes Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. 2018. Automatic Differentiation in Machine Learning: a Survey. Journal of Machine Learning Research 19, 18, 153 (2018), 1–43. http://jmlr.org/papers/v18/17-468.html
Jan Bender, Matthias Müller, Miguel A Otaduy, Matthias Teschner, and Miles Macklin. 2014. A survey on position-based simulation methods in computer graphics. In Computer graphics forum, Vol. 33. Wiley Online Library, 228–251.

ACM Trans. Graph., Vol. 40, No. 4, Article 79. Publication date: August 2021.
