
Article:
Shen, S, Yang, Y, Shao, T et al. (2021) High-order Differentiable Autoencoder for Nonlinear Model Reduction. ACM Transactions on Graphics, 40 (4). 68. ISSN: 0730-0301

https://doi.org/10.1145/3450626.3459754

High-order Differentiable Autoencoder for Nonlinear Model Reduction

SIYUAN SHEN, State Key Lab of CAD&CG, Zhejiang University, China
YANG YIN, Clemson University, USA
TIANJIA SHAO, State Key Lab of CAD&CG, Zhejiang University, China
HE WANG, University of Leeds, United Kingdom
CHENFANFU JIANG, University of Pennsylvania, USA
LEI LAN, Clemson University, USA
KUN ZHOU, State Key Lab of CAD&CG, Zhejiang University, China

Fig. 1. In this paper, we exploit deep autoencoder (DAE) networks to accelerate physics-based simulation. In order to model nonlinear subspace dynamics accurately, second- and high-order derivatives of the deep decoder net must be efficiently evaluated to match the subspace simulation frame rate. We address this technical challenge by collectively applying complex-step network perturbations to the deep net. This is the first time a high-order differentiable neural net is employed in physical simulation problems. Our method can be further strengthened with the domain decomposition method as a nonlinear DAE better captures local deformation effects. In this example, the puffer ball has 320 elastic strings, and we assign a $n_p = 10$ linear subspace and a $n_q = 5$ nonlinear subspace at each string. With the help of substructured deformation, DAE-based nonlinear reduction produces interesting animation effects. We believe this is a representative example showing case the advantage of data-driven animation using DAE. As the geometries of all the strings are the same, generating local training poses is more effective.

This paper provides a new avenue for exploiting deep neural networks to improve physics-based simulation. Specifically, we integrate the classic Lagrangian mechanics with a deep autoencoder to accelerate elastic simulation of deformable solids. Due to the inertia effect, the dynamic equilibrium cannot be established without evaluating the second-order derivatives of the deep autoencoder network. This is beyond the capability of off-the-shelf automatic differentiation packages and algorithms, which mainly focus on the gradient evaluation. Solving the nonlinear force equilibrium is even more challenging if the standard Newton's method is to be used. This is because we need to compute a third-order derivative of the network to obtain the variational Hessian. We attack those difficulties by exploiting complex-step finite difference, coupled with reverse automatic differentiation. This strategy allows us to enjoy the convenience and accuracy of complex-step finite difference and in the meantime, to deploy complex-value perturbations as collectively as possible to save excessive network passes. With a GPU-based implementation, we are able to wield deep autoencoders (e.g., 10+ layers) with a relatively high-dimension latent space in real-time. Along this pipeline, we also design a sampling network and a weighting network to enable *weight-varying* Cubature integration in order to incorporate nonlinearity in the model reduction. We believe this work will inspire and benefit future research efforts in nonlinearly reduced physical simulation problems.

Authors' addresses: Siyuan Shen, State Key Lab of CAD&CG, Zhejiang University, 866 Yuhangtang Rd, Hangzhou, 310058, China, shensiyuan@zju.edu.cn; Yang Yin, Clemson University, USA, yin5@clemson.edu; Tianjia Shao, State Key Lab of CAD&CG, Zhejiang University, China, tianjiashao@gmail.com; He Wang, University of Leeds, United Kingdom, H.E.Wang@leeds.ac.uk; Chenfanfu Jiang, University of Pennsylvania, USA, cffjiang@seas.upenn.edu; Lei Lan, Clemson University, USA, lanlei.virhum@gmail.com; Kun Zhou, State Key Lab of CAD&CG, Zhejiang University, China, kunzhou@acm.org.

CCS Concepts: • **Computing methodologies** → **Physical simulation**; **Dimensionality reduction and manifold learning**.

Additional Key Words and Phrases: Model reduction, Autoencoder, Differentiation, GPU, Deformable model

Reference Format:
Siyuan Shen, Yang Yin, Tianjia Shao, He Wang, Chenfanfu Jiang, Lei Lan, and Kun Zhou. 2021. High-order Differentiable Autoencoder for Nonlinear Model Reduction. arXiv (February 2021), 15 pages.

1 INTRODUCTION

Model reduction is a widely-used and highly-effective technique for accelerating physically-based simulation. It is also sometimes known as reduced-order simulation or subspace simulation. While named differently, the core idea is to build a linear subspace with reduced degrees-of-freedom (DOFs) so that the physical equations can be solved with a system of a smaller size. This approach is sensible

because many parts of the real physical world evolve smoothly and continuously along the time and space. Sharp and high-frequency physical changes are less common and should be treated with dedicated numerical methods. Existing model reduction methods have been dominantly *linear* reduction with a constant tangent space. The expressivity of linear reduction is a known limitation. As many physical phenomena are intrinsically nonlinear, a linearly reduced model only covers a small fraction of the dynamics space – all the information outside of the subspace is filtered. Thus, one has to (substantially) increase the dimensionality of the subspace to incorporate a desired nonlinear effect even this effect itself may be of low rank (thinking of a bead travelling on the circle).

One question rises naturally: can we build a nonlinear reduction framework with a time-varying (as opposed to constant) tangent space that best fits "local" dynamics? The challenges are twofold. First, the underlying manifold representing the nonlinear dynamics is often too complex to be expressed in a closed form. Developing a nonlinear subspace-fullspace transformation, as a counterpart of modal analysis in linear reduction, is theoretically difficult. Second, a nonlinear reduction brings extra computation burdens to the simulation, largely originating from the need for evaluating the derivatives of the subspace-fullspace transformation function. The computational cost goes up quickly with respect to the subspace size, which in turn neutralizes the original motivation of applying model reduction.

In this paper, we propose a new nonlinear model reduction framework that is tightly coupled with the classic Lagrangian mechanics. Although we demonstrate the effectiveness of our method in the context of elastic simulation, we believe our method could also be useful in other physics-based simulation problems like fluid [Kim and Delaney 2013] or cloth animations [Hahn et al. 2014]. Our reduction mechanism is data-driven with a deep autoencoder (DAE) in the loop, which obviates the need for a closed-form subspace-fullspace transformation function. DAE is an unsupervised learning architecture skilled in data compression [Hinton and Salakhutdinov 2006] and has been proven effective in deformable simulation recently [Fulton et al. 2019]. Along this direction, we augment the DAE network with the complex-step finite difference (CSFD) method [Martins et al. 2003], enabling its high-order differentiability, so that the neural net can be computationally integrated with Lagrangian formulation. To achieve this goal, we make several mentionable technical contributions:

• *Efficient high-order differentiable deep autoencoder.*
A physically accurate coupling between DAE and elastic dynamics requires the information of the first- and second-order derivatives (for Newtonian equations of motion). Existing differentiation techniques such as backpropagation (BP) [Hecht-Nielsen 1992] for neural networks or automatic differentiation (AD) [Bücker et al. 2006] for more general computations are optimized for gradient estimation only, and become cumbersome in high-order cases. We leverage CSFD to facilitate the differentiation of the encoder network. Conceptually straightforward, this however is not "as easy as pie" as it appears. Albeit the excellent numerical robustness and accuracy, CSFD needs to apply a complex-value perturbation for each input variable, leading to excessive forward passes of the deep neural net. We resolve this challenge by applying the function perturbation *collectively* and deploying CSFD *inside* other differentiation procedures such as BP and directional derivative. With a GPU-based implementation, we simulate complex nonlinear models in real time with a deep decoder net in the loop.

• *Coupling PCA with deep encoding net.*
The tangent space of a DAE may vary drastically to incorporate nonlinearity seen in the training poses yielding a bumpy and uneven deformation manifold. This is analogous to over-fitting. A possible cure is to use contractive autoencoder or CAE [Rifai et al. 2011b]. CAE adds a regularizer in the objective function that forces the network to learn a function that is robust under slight input variations. While it could be a viable solution, we propose a more convenient and effective option. In our framework, DAE is constructed within the residual space of a standard PCA. In other words, DAE is designed to be complementary to an underlying linear subspace, and the latter guarantees the existence of a smooth tangent variation. With this design, we can deepen the DAE architecture to capture nonlinear and salient deformation poses.

• *Weight-varying subspace integration using deep neural networks.*
Reduced simulation is often coupled with sparse force and Hessian integration, which down samples element-wise volumetric integration to a small collection of key elements, called Cubature elements [An et al. 2008]. After Cubature elements are selected, one also needs to compute its integration weight by solving a non-negative least-square problem. The weight coefficient of a Cubature element is typically fixed given the training set. This is reasonable for linear reduction and works well in practice. However, in the context of nonlinear reduction, as the tangent space varies along the simulation, fixed weighting Cubature is problematic. To this end, we propose a deep neural network (DNN) based sampling method, that fully replaces Cubature training. Our DNN has two modules. The first module is a graph convolution network (GCN) that outputs the possibility of an element being a Cubature element. On the top of it, the second module is a DNN, which predicts the weight of selected Cubature elements. The last layer of this DNN carries out a per-neuron square operation to ensure the final network output is non-negative. The training alternates between those two modules. Unlike conventional Cubature sampling strategy, our network-based approach is able to select multiple Cubature elements each iteration, thus greatly shortens the training time.

We have evaluated our framework on various simulation scenarios, and our method produces visually-plausible results in real time or at an interactive rate. We also notice that our nonlinear model reduction framework synergizes with domain decomposition method [Barbič and Zhao 2011; Wu et al. 2015; Yang et al. 2013] – a small-size nonlinear subspace captures deformation effects much better at a local domain than over the entire deformable body. To this end, we also demonstrate examples combining DAE and domain decomposition. As a natural follow up of our method, we do not intend to over claim this extension as our contribution. Model reduction, regardless nonlinear or linear, seeks for smart trade-offs among simulation effects, accuracy, and performance. Arguing conclusively that the nonlinear model reduction is always better than linear model reduction techniques is too bold and over-confident to

us. Indeed, one should scrutinize various aspects in practice, such as the problem size, expected results, time budget, hardware resources etc. before choosing a specific simulation algorithm. Regardless, we do believe the techniques purposed in the paper are worthy and non-trivially advance state-of-the-art model reduction methods.

2 RELATED WORK

Model reduction has been successfully employed in many simulation-related problems in computer graphics including fluid dynamics [Kim and Delaney 2013; Treuille et al. 2006], cloth animation [Hahn et al. 2014], shape deformation [Von-Tycowicz et al. 2015; Wang et al. 2015], material design [Musialski et al. 2016; Xu et al. 2015], animation control [Barbič et al. 2009, 2012] etc. In this paper, we narrow our focus on using data-driven nonlinear model reduction to improve elastic simulation of solid objects.

There are several well-established numerical solutions for deformable models such as finite element method (FEM) [Zienkiewicz et al. 1977], finite difference method [Zhu et al. 2010], meshless method [Martin et al. 2010; Müller et al. 2005], or mass-spring system [Liu et al. 2013]. Most of them end up with solving a large-scale nonlinear system if an implicit time integration scheme is used. For high-resolution models, computing their time-varying nonlinear dynamics is expensive. Speeding up the deformable simulation can be achieved using carefully designed numerical treatments like the multigrid method [Tamstorf et al. 2015; Zhu et al. 2010], delayed matrix update [Hecht et al. 2012], or parallelizable solvers [Fratarcangeli et al. 2016; Wang and Yang 2016]. These methods focus on improving the performance for the *fullspace* nonlinear optimization without condensing the simulation scale.

Acceleration can also be achieved using model reduction, which removes less important DOFs and creates a subspace representation of fullspace DOFs. Modal analysis [Choi and Ko 2005; Hauser et al. 2003; Pentland and Williams 1989] and its first-order derivatives [Barbič and James 2005; Yang et al. 2015] are often considered as the most effective way for the subspace construction. Displacement vectors from recent fullspace simulations can also be utilized as subspace bases [Kim and James 2009]. Alternatively, it is also viable to coarsen geometric representation to prescribe the dynamics of a fine model like skin rigging, a technique widely used in animation systems [James and Twigg 2005]. Analogously, Capell and colleagues [2002] deformed an elastic body using an embedded skeleton; Gilles and colleagues [2011] used rigid frames to drive the deformable simulation; Faure and colleagues [2011] used scattered *handles* for reduced models; Lei and colleagues [2020] combined model reduction with collision processing using medial axis transform; Martin and colleagues [2010] used sparsely-distributed integrators named *elastons* to model the nonlinear dynamics of rod, shell, and solid uniformly.

Those prior arts demonstrate impressive results, often with orders-of-magnitude performance speedups with model reduction. In most cases, the generalized coordinate linearly depends on the fullspace displacement in the form of $\mathbf{u} = \mathbf{U}\mathbf{q}$ with $\mathbf{U}$ being constant. This is why we refer to them as linear subspace methods. On the contrary, nonlinear reduction holds a more complicated relation between generalized and fullspace coordinates. For instance, nonlinear modal analysis [Pesheck et al. 2001] aims to extend its linear version, and it has been used in structural analysis [Setio et al. 1992]. However, it is hardly useful for simulation acceleration – extracting the modal space for a given configuration is normally dealt with by solving an eigenproblem (i.e., as in linear modal analysis), and it is clearly infeasible to exhaustively sample all the system configurations even in the pre-computation stage. Only when the animation follows some pre-known patterns, we may re-use the solution of rest-shape eigenproblem [Mukherjee et al. 2016] or interpolate multiple sparsely chosen linear subspaces [Xu and Barbič 2016]. Due to this challenge, the nonlinear subspace method is less explored.

In this paper, we do not aim to derive a closed-form mathematical formulation connecting the generalized coordinate and the fullspace coordinate. Instead, we leave this challenge to a deep neural network that learns the map directly from many seen simulation poses. This is a straightforward data-driven approach and has been exploited in graphics for years [Ladickỳ et al. 2015; Wang et al. 2011]. Our novelty however is to *enable its efficient and high-order differentiability so that the DNN can be embedded into the classic physical simulation frameworks* such as Lagrangian mechanics [Brizard 2014] etc. The DNN used in our framework is a deep autoencoder or DAE [Hinton and Salakhutdinov 2006] originally designed for dimension reduction. Its superior performance in data compression and multidimensional scaling has quickly drawn many attentions. DAE is successfully deployed in NLP [Socher et al. 2011], image/video compression [Ballé et al. 2016; Habibian et al. 2019], GAN [Makhzani et al. 2015], facial recognition [Zeng et al. 2018], 3D shape analysis [Nair and Hinton 2009], just to name a few. The volume of DAE-related studies is too vast to be contained here.

The most relevant study of our work is the contribution from Fulton and colleagues [2019]. Indeed, we are strongly motivated and inspired by those recent efforts [Fulton et al. 2019; Wiewel et al. 2019] that also seek for DAE-based nonlinear reduction. To this end, we re-examine each step along the pipeline of reduced simulation and devise a comprehensive solution to couple DAE with nonlinear elastic simulation seamlessly. One core ingredient is the high-order differentiability that should be evaluated efficiently to match the frame rate of model reduction. We deliver this important technical asset by leveraging complex-step finite difference or CSFD [Martins et al. 2003]. Similar to standard finite difference, CSFD applies a perturbation to the function input and evaluates how the perturbation alters the function output. This perturbation however, is a complex-value quantity in CSFD, which avoids the numerical issue of subtractive cancellation. While CSFD seems to be a possible approach to the differentiability of DAE, its efficient deployment for high-order differentiation remains challenging in practice. An naïve implementation of CSFD requires one forward pass of the net for each input variable. This scheme leads to $\mathbf{O}(n^3)$ passes for a DAE-enabled Newton iteration. As discussed in § 4, we attack this difficulty by applying the complex stepping in CSFD collectively whenever possible to remove redundant network passes. This method allows us to efficiently run the DAE differentiation on GPU and obtain its high-order derivatives in milliseconds. In order to ensure the smoothness of the simulation tangent space, our framework consists of two layers: the first layer is a standard PCA-based linear subspace and within the orthogonal space of which,

DAE is deployed to capture nonlinear deformations more effectively. Lastly, we also design a DNN-based Cubature training procedure to generate pose-dependent weight coefficients for a more accurate subspace integration.

3 LINEAR AND NONLINEAR MODEL REDUCTION

To make the paper more self-contained, we start with a brief review of the linear model reduction framework, and show its nonlinear generalization with DAE afterwards. Here, we assume that a DAE is differentiable and defer the discussion about how to compute its first- and high-order derivatives to the next section.

3.1 Linear Model Reduction

Under the FEM discretization, the motion of an elastically deformable solid can be described with the Euler-Lagrange equation:

$$\mathbf{M}\ddot{\mathbf{u}} + \mathbf{f}_{damp}(\mathbf{u}, \dot{\mathbf{u}}) + \mathbf{f}_{int}(\mathbf{u}) = \mathbf{f}_{ext}, \tag{1}$$

where $\mathbf{M} \in \mathbb{R}^{N\times N}$ is the fullspace mass matrix; $\mathbf{f}_{int}$ and $\mathbf{f}_{ext}$ are the nonlinear internal force and external force. Here, we lump $\mathbf{M}$ to be a diagonal matrix. $\mathbf{f}_{damp}$ is the damping force, and it is often modeled, under the assumption of Rayleigh damping, as:

$$\mathbf{f}_{damp} = \left(\alpha\mathbf{M} + \beta\frac{\partial \mathbf{f}_{int}(\mathbf{u})}{\partial \mathbf{u}}\right)\dot{\mathbf{u}}. \tag{2}$$

Eq. (1) describes the force equilibrium at all N DOFs of the unknown displacement vector $\mathbf{u}$. Computing $\mathbf{u}$ in Eq. (1) using nonlinear methods like Newton's method needs to solve an N-by-N linearized system repeatedly, which is slow and expensive for large-scale models. Linear model reduction prescribes the kinematic of this N-dimension system with a set of generalized coordinate $\mathbf{p}$ such that $\mathbf{u} = \mathbf{U}\mathbf{p}$. $\mathbf{U} \in \mathbb{R}^{N\times n}$ is sometimes called subspace matrix, which is constant in linear reduction. An important convenience brought by the linearity is the time derivatives of $\dot{\mathbf{u}} = \mathbf{U}\dot{\mathbf{p}}$ and $\ddot{\mathbf{u}} = \mathbf{U}\ddot{\mathbf{p}}$ follow the same relation. Therefore, Eq. (1) can be projected into the column space of $\mathbf{U}$ as:

$$\mathbf{M}_p\ddot{\mathbf{p}} + \mathbf{U}^\top\mathbf{f}_{damp}(\mathbf{U}\mathbf{p}, \mathbf{U}\dot{\mathbf{p}}) + \mathbf{U}^\top\mathbf{f}_{int}(\mathbf{U}\mathbf{p}) = \mathbf{U}^\top\mathbf{f}_{ext}, \tag{3}$$

where $\mathbf{M}_p = \mathbf{U}^\top\mathbf{M}\mathbf{U}$ is the reduced mass matrix. Eq. (3) has the same structure of Eq. (1) despite under a more compact representation of $\mathbf{p}$.

3.2 Nonlinear Model Reduction

Nonlinear reduction also uses a set of generalized coordinates $\mathbf{q}$. However, the relation between $\mathbf{u}$ and $\mathbf{q}$ is in a more generic form of $\mathbf{u} = D(\mathbf{q})$. Intuitively, D prescribes an n-dimension *deformation manifold* embedded in $\mathbb{R}^N$. Applying time differentiation at both side yields:

$$\dot{\mathbf{u}} = \frac{\mathrm{d}D(\mathbf{q})}{\mathrm{d}t} = \frac{\partial D(\mathbf{q})}{\partial \mathbf{q}}\dot{\mathbf{q}} = \mathbf{J}\dot{\mathbf{q}}, \tag{4}$$

and

$$\ddot{\mathbf{u}} = \frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial D(\mathbf{q})}{\partial \mathbf{q}}\dot{\mathbf{q}}\right) = (\mathcal{H}\cdot\dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{J}\ddot{\mathbf{q}}, \tag{5}$$

where $\mathbf{J} = \partial D(\mathbf{q})/\partial\mathbf{q} \in \mathbb{R}^{N\times n}$ is the Jacobian of $D(\mathbf{q})$, which depends on $\mathbf{q}$ and spans the *tangent space* at a given reduced coordinate. $\mathcal{H} \in \mathbb{R}^{N\times n\times n}$ is a third tensor of the second-order derivative (i.e., Hessian) of D. Substituting Eqs. (4) and (5) into Eq. (1) followed by a tangent space projection gives the nonlinearly reduced equation of motion:

$$\mathbf{J}^\top\mathbf{M}\big((\mathcal{H}\cdot\dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{J}\ddot{\mathbf{q}}\big) + \mathbf{J}^\top\mathbf{f}_{int}(D(\mathbf{q})) = \mathbf{J}^\top\mathbf{f}_{ext}. \tag{6}$$

Here, the damping force term is ignored for a more concise notation. We note that if $D(\mathbf{q})$ is linear, $\mathcal{H}$ vanishes and $\mathbf{J} = \mathbf{U}$. Eq. (6) echoes Eq. (3) completely.

Given a time integration algorithm on $\mathbf{q}$ e.g., the implicit Euler method, we have: $\mathbf{q} = \bar{\mathbf{q}} + h\dot{\mathbf{q}}$ and $\dot{\mathbf{q}} = \bar{\dot{\mathbf{q}}} + h\ddot{\mathbf{q}}$, where h is the time step size, and $\bar{(\cdot)}$ indicates the kinematic variable is from the previous time step. The final system that needs to be solved becomes:

$$\mathbf{J}^\top\mathbf{M}\mathbf{J}(\mathbf{q} - \bar{\mathbf{q}} - h\bar{\dot{\mathbf{q}}}) + \mathbf{J}^\top\mathbf{f}_{fict}(\mathbf{q}) + h^2\mathbf{J}^\top\mathbf{f}_{int}(\mathbf{q}) = h^2\mathbf{J}^\top\mathbf{f}_{ext}, \tag{7}$$

with

$$\mathbf{f}_{fict} = \mathbf{M}\left([\mathcal{H}\cdot(\mathbf{q} - \bar{\mathbf{q}})]\,(\mathbf{q} - \bar{\mathbf{q}})\right). \tag{8}$$

$\mathbf{f}_{fict}$ is the fictitious force that responds for inertia effects associated with the varying Jacobian $\mathbf{J}$. We also consider $\mathbf{M}_q = \mathbf{J}^\top\mathbf{M}\mathbf{J}$ is the reduced mass matrix of the nonlinear reduction, which is no longer constant as $\mathbf{J}$ also depends on $\mathbf{q}$.

3.3 A Quick Discussion

Clearly, $\mathbf{f}_{fict}$ is the most tricky part in Eq. (7). $\mathcal{H}$ is the Hessian of the coordinate transformation D. Not only a third tensor, but $\mathcal{H}(\mathbf{q})$ is also a function of $\mathbf{q}$. Therefore, if we want to solve Eq. (7) using, for instance, Newton's method in the implicit integration, we need to compute $\partial\mathcal{H}/\partial\mathbf{q}$ to assemble the corresponding system matrix, which is a forth tensor and the resultant of third-order differentiation over D. This nasty computation stands as a major obstacle for nonlinear model reduction. In [Fulton et al. 2019], the fictitious force term is discarded in the time integration of the generalized coordinate. This heuristic can somewhat be understood as performing an *explicit* subspace projection at the current time step ignoring the fact that a generalized velocity $\dot{\mathbf{q}}$ also brings inertia effects when D is nonlinear.

On the one hand, we consider ignoring $\mathbf{f}_{fict}$ reasonable and an understandable compromise in the setting of [Fulton et al. 2019]. First of all, $\mathbf{f}_{fict}$ vanishes under quasi-static deformations as $\dot{\mathbf{q}}$ is close to zero. Secondly, in [Fulton et al. 2019] the encoding-decoding network is shallow, and D represents a net of only two layers. In addition, a preliminary PCA is performed to "regularize" raw training poses. Those treatments effectively suppress the nonlinearity in $\mathbf{J}$ (so $\mathcal{H}$ is small) and lessen the inertia deformation induced by $\mathbf{f}_{fict}$. On the other hand, as $\mathbf{f}_{fict}$ is missed in [Fulton et al. 2019], the underlying dynamic equation is inaccurate anyway. Visible artifacts are inevitable under higher-velocity deformations or a deeper DAE is employed (i.e., in our case).

3.4 PCA-orthogonal DAE Reduction

An autoencoder is an unsupervised learning algorithm that condenses the input high-dimension data into a low-dimension *latent space* (i.e., encoding), which is then expanded to the original dimensionality to monitor the compression loss (i.e., decoding). If this network only has one hidden layer, or it does not involve nonlinear activations, the autoencoder is similar to PCA [Bourlard and Kamp 1988]. In this case D is linear, and the resulting network after training spans the same linear subspace as PCA does (under L2 loss).

This however is, not what we seek for in nonlinear model reduction since D is expected to capture as much nonlinearity as possible to enrich the subspace expressivity. To this end, we ought to keep the autoencoder deep and nonlinear. Unfortunately, too much nonlinearity seems to be harmful to the simulation as well. As illustrated in Fig. 2, with increased nonlinearity, the network (i.e., the curve in the figure) can be stretched to reach some irregular and distant poses in the training set. In the meantime, the geometry of the deformation manifold also becomes more wiggling – same as what we experience in high-order polynomial fitting. As we know, the simulation under nonlinear reduction corresponds to travelling on the deformation manifold of D, driven by the generalized forces in the tangent space. A wiggling manifold could stiffen the simulation and induce artifacts.

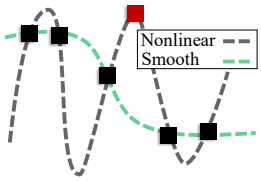

Fig. 2. Increased nonlinearity better fits training poses but also makes the network more wiggling.

There are several possible remedies of this issue. As in [Fulton et al. 2019], one could regularize the training data before the network training. This strategy is commonly used in training deep neural nets on a very large-scale data set that could potentially be noisy e.g., ImageNet [Russakovsky et al. 2015]. However, the training data in our case are synthesized by running physical simulations, and they are *noise-free*. While PCA regularization certainly prevents overfitting, it also negatively impacts the richness of the nonlinear subspace. Alternatively, CAE is also a promising method [Rifai et al. 2011a]. It injects a penalty term related to $\|\mathbf{J}\|_F$ to enhance the smoothness of D so that a highly curved manifold is unlikely. Unfortunately, neither method looks attractive to us: the very reason of using DAE is to enhance the nonlinearity of the subspace, while both PCA regularization and Jacobian penalty aim to prune the subspace nonlinearity, contradicting our original motivation. The remaining option is to expand the dimension of the latent space, which is also problematic knowing that the computational cost for nonlinear model reduction is much higher than linear reduction (i.e., due to the evaluation of high-order differentiation). If the majority information encoded in the DAE is close to linear, why bother using nonlinear reduction and a deep network at the very beginning?

Our answer to this dilemma is to split the total simulation space into two *orthogonal* spectra: a PCA-based linear subspace (or it could be constructed by any linear model reduction methods) $\mathcal{S}_p$ and a DAE-based nonlinear manifold $\mathcal{S}_q$ such that $\mathcal{S}_p \perp \mathcal{S}_q$. The orthogonality allows the dynamics from both subspaces to be simply super-positioned. The advantages of this subspace design are multifaceted. First of all, we can now increase the dimension of the linear subspace with a moderate cost to the overall simulation performance. Secondly, under this design, PCA basis matrix is part of the Jacobian of the overall subspace $\mathcal{S}_p \cup \mathcal{S}_q$. Therefore, the simulation does not experience the locking artifact. Explicitly building the linear subspace also allows us to push the depth of the DAE as needed to capture nonlinear deformations and keep latent space highly compact at the same time. Lastly, we note that such multi-layer subspace construction has been successfully employed in simulation [Harmon and Zorin 2013; Zhang et al. 2020], but first time in conjunction with a deep network.

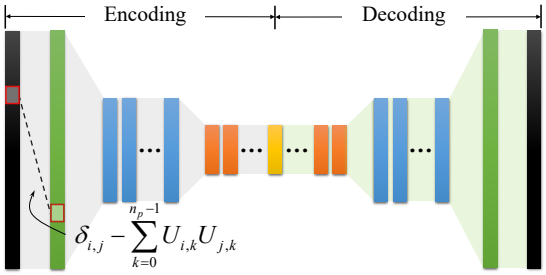

Fig. 3. The network structure of our DAE. At both sides of the DAE, we append a fully connected filtering layer (in green) to remove any displacements from PCA space $\mathcal{S}_p$.

3.5 Network Architecture

The network architecture of our DAE is visualized in Fig. 3. It has a symmetric structure at encoding and decoding parts. Before training the DAE, we perform PCA over the training set to obtain basis vectors of $\mathcal{S}_p$. They are packed into the matrix $\mathbf{U}$. $\mathbf{U}$ is $N \times n_p$, where n_p represents the dimensionality of $\mathcal{S}_p$. Columns in $\mathbf{U}$ are all unit vectors, and they are orthogonal to each other. To ensure $\mathcal{S}_p \perp \mathcal{S}_q$, we append a *filtering* layer at both ends of the encoder and and the decoder. This filtering layer is fully connected (FC) and has fixed weights: the weight coefficient of the edge connecting i-th and j-th neurons before and after this FC layer is $\delta_{i,j} - \sum_{k=0}^{n_p-1} U_{i,k}U_{j,k}$, where $\delta_{i,j} = 1$ for $i = j$ and 0 otherwise. In fact, this FC layer carries out a matrix-vector product of $(\mathbf{I} - \mathbf{U}\mathbf{U}^\top)\mathbf{x}$ for an input vector $\mathbf{x}$, which removes any components in $\mathbf{x}$ that generate non-zero projections in $\mathcal{S}_p$ so that the input of the encoder and the output from the decoder are all orthogonal to $\mathcal{S}_p$.

After filtering, the DAE moves to an intermediate activation part, which consists of multiple (e.g., 6 to 8) FC layers of the same width. The width is normally set at the order of $\log N$. Each layer is nonlinearly activated. Our activation function is quite different from other deep nets. ReLU (rectified linear unit) is a widely chosen activation, and works well in many deep learning tasks by default [Nair and Hinton 2010]. However, as ReLU is a linear activation, it fails to nonlinearly compress the training data. More importantly, ReLU is C^0 continuous, and a DAE only activated by ReLU may degenerate to PCA. The exponential linear unit or ELU enhances the smoothness of ReLU, but it could remain a linear activator for certain input signals. To this end, we use the *trigonometric function* $\sin x$ as our activation. $\sin x$ has a derivative of an arbitrary order, and it does not have a saturated gradient at both directions. This pleasing property frees us from worrying about the vanishing gradient problem [Hochreiter et al. 2001] even the network is deep (over 10 layers). Finally, the feature vector is compressed to the latent space. We use n_q to denote the dimension of $\mathcal{S}_q$. n_q is a small number, typically below a couple of dozens in our experiments.

3.6 The Simulation System

Now, we have everything to give the formulation of the final system we need to solve. With $\mathcal{S}_p$ and $\mathcal{S}_q$ constructed, the fullspace displacement is written as:

$$\mathbf{u} = \mathbf{U}\mathbf{p} + D(\mathbf{q}), \quad \text{s.t.} \quad \mathbf{U}^\top D(\mathbf{q}) = 0. \tag{9}$$

Thanks to the orthogonality between $\mathcal{S}_p$ and $\mathcal{S}_q$, we stack Eqs. (3) and (6) jointly to obtain:

$$\widetilde{\mathbf{J}}^\top \mathbf{M}\widetilde{\mathbf{J}}\left(\mathbf{r} - \bar{\mathbf{r}} - h\bar{\dot{\mathbf{r}}}\right) + \widetilde{\mathbf{J}}^\top \mathbf{f}_{fict} + h^2\widetilde{\mathbf{J}}^\top\left(\mathbf{f}_{int} - \mathbf{f}_{ext}\right) = 0, \tag{10}$$

where $\mathbf{r} = [\mathbf{p}^\top, \mathbf{q}^\top]^\top$ is the generalized coordinate concatenating both $\mathbf{p}$ and $\mathbf{q}$, and $\widetilde{\mathbf{J}} = [\mathbf{U}, \mathbf{J}] \in \mathbb{R}^{N\times(n_p+n_q)}$. Here, $\mathbf{f}_{fict}$ is in the same form of Eq. (8) because it vanishes in $\mathcal{S}_p$. Eq. (10) can then be concisely written as $\phi(\mathbf{r}) = 0$. Its Jacobian is a $(n_p + n_q) \times (n_p + n_q)$ matrix:

$$\begin{aligned}\frac{\partial \phi}{\partial \mathbf{r}} = \widetilde{\mathcal{H}}^\top \mathbf{M}\widetilde{\mathbf{J}}\left(\mathbf{r} - \bar{\mathbf{r}} - h\bar{\dot{\mathbf{r}}}\right) + \widetilde{\mathcal{H}}^\top \mathbf{f}_{fict} + \widetilde{\mathbf{J}}^\top \mathbf{M}\left[\mathbf{U}, \Delta\mathbf{J} + \mathbf{J}\right] \\ - h^2\widetilde{\mathcal{H}}^\top\left(\mathbf{f}_{int} - \mathbf{f}_{ext}\right) + h^2\left(\widetilde{\mathbf{J}}^\top \frac{\partial \mathbf{f}_{int}}{\partial \mathbf{u}}\widetilde{\mathbf{J}}\right),\end{aligned} \tag{11}$$

which needs to be updated and solved at each Newton iteration. Here, $\widetilde{\mathcal{H}} = \partial^2\mathbf{u}/\partial\mathbf{r}^2 = \mathtt{diag}(0, \mathcal{H})$. The most involving term is $\Delta\mathbf{J}$, which is defined as:

$$\Delta\mathbf{J} \triangleq (\mathbb{S} \cdot (\mathbf{q} - \bar{\mathbf{q}})) \cdot (\mathbf{q} - \bar{\mathbf{q}}) + \mathcal{H} \cdot (3\mathbf{q} - 3\bar{\mathbf{q}} - h\bar{\dot{\mathbf{q}}}). \tag{12}$$

In order to compute $\Delta\mathbf{J}$, we need to calculate $\mathbb{S}$, an $N \times n_q \times n_q \times n_q$ forth tensor, and it is the third-order derivative of DAE: $\partial^3 D(\mathbf{q})/\partial\mathbf{q}^3$. If we choose to use first-order or quasi-Newton solvers [Liu et al. 2017], the computation of $\mathbb{S}$ could be avoided, but we still need to compute the Hessian $\mathcal{H}$. Nevertheless, for subspace simulation with a small-size system matrix, Newton's method with a direct linear solver like Cholesky is always preferred.

Evaluating high-order differentiation of a deep net is not intuitive. Currently, gradient-based optimization is the mainstream solution for the network training, where the network gradient can be computed via BP. Second- and high-order derivatives are not well supported and are not efficient enough for subspace simulation tasks. Next, we discuss how we solve this technical challenge by exploiting the complex-step finite difference scheme.

4 HIGH-ORDER DIFFERENTIABILITY VIA CSFD

Computing the derivative of a function is omnipresent in physics-based simulation. It is typically done by inferring analytic form of the derivative function by hand or with assistance from some *symbolic differentiation* software like `Mathematica` [Wolfram et al. 1999]. Alternatively, it is also possible to approximate the derivative numerically. The finite difference is the most commonly-used method, which applies a small perturbation h to the function input and the first-order function derivative can be estimated as: $f'(x) \approx (f(x + h) - f(x))/h$. However, it is also known finite difference suffers with the numerical stability issue named *subtractive cancellation* [Luo et al. 2019]. This limitation could be avoided by complex-step finite difference or CSFD [Luo et al. 2019; Martins et al. 2003].

4.1 First- and High-order CSFD

Let $(\cdot)^*$ denote a complex variable, and suppose $f^* : \mathbb{C} \to \mathbb{C}$ is differentiable around $x_0^* = x_0 + 0i$. With an imaginary perturbation hi, f^* can be expanded as:

$$f^*(x_0 + hi) = f^*(x_0) + f^{*'}(x_0) \cdot hi + \mathbf{O}(h^2). \tag{13}$$

We can "promote" a real-value function f to be a complex-value one f^* by allowing complex inputs while retaining its original computation procedure. Under this circumstance, we have $f^*(x_0) = f(x_0), f^{*'}(x_0) = f'(x_0) \in \mathbb{R}$. Extracting imaginary parts of both sides in Eq. (13) yields:

$$\mathtt{Im}\big(f^*(x_0 + hi)\big) = \mathtt{Im}\big(f^*(x_0) + f^{*'}(x_0) \cdot hi\big) + \mathbf{O}(h^3). \tag{14}$$

Note that the error term ($\mathbf{O}(h^3)$) in Eq. (14) is cubic because the quadratic term of h in Eq. (13) is a real quantity and is excluded by `Im` operator. We then have the first-order CSFD approximation:

$$f'(x_0) = \frac{\mathtt{Im}\big(f^*(x_0 + hi)\big)}{h} + \mathbf{O}(h^2) \approx \frac{\mathtt{Im}\big(f^*(x_0 + hi)\big)}{h}. \tag{15}$$

It is clear that Eq. (15) does not have a subtractive numerator, meaning it only has the round-off error regardless of the size of the perturbation h. If $h \sim \sqrt{\epsilon}$ i.e., around 1×10^{-16}, CSFD approximation error is at the order of the machine epsilon ϵ. Hence, CSFD can be as accurate as analytic derivative because the analytic derivative also has a round-off error of ϵ.

The generalization of CSFD to second- or even higher-order differentiation is straightforward by making the perturbation a *multi-complex* quantity [Lantoine et al. 2012; Nasir 2013]. The multicomplex number is defined recursively: its base cases are the real set $\mathbb{C}^0 = \mathbb{R}$, and the regular complex set $\mathbb{C}^1 = \mathbb{C}$. $\mathbb{C}^1$ extends the real set ($\mathbb{C}^0$) by adding an imaginary unit i as: $\mathbb{C}^1 = \{x + yi | x, y \in \mathbb{C}^0\}$. The multicomplex number up to an order of n is defined as: $\mathbb{C}^n = \{z_1 + z_2 i_n | z_1, z_2 \in \mathbb{C}^{n-1}\}$. Under this generalization, the multicomplex Taylor expansion becomes:

$$\begin{aligned}f^\star(x_0 + hi_1 + \cdots + hi_n) = f^\star(x_0) + f^{\star'}(x_0)h\sum_{j=1}^{n} i_j \\ + \frac{f^{\star''}(x_0)}{2}h^2\Big(\sum_{j=1}^{n} i_j\Big)^2 + \cdots \frac{f^{\star(k)}}{k!}h^k\Big(\sum_{j=1}^{n} i_j\Big)^k \cdots\end{aligned} \tag{16}$$

Here, $\left(\sum i_j\right)^k$ can be computed following the *multinomial theorem*, and it contains products of mixed k imaginary directions for k-th-order terms. For instance, the second-order CSFD formulation can then be derived as follows:

$$\begin{cases}\dfrac{\partial^2 f(x,y)}{\partial x^2} \approx \dfrac{\mathtt{Im}^{(2)}\left(f(x + hi_1 + hi_2, y)\right)}{h^2} \\ \dfrac{\partial^2 f(x,y)}{\partial y^2} \approx \dfrac{\mathtt{Im}^{(2)}\left(f(x, y + hi_1 + hi_2)\right)}{h^2} \\ \dfrac{\partial^2 f(x,y)}{\partial x \partial y} \approx \dfrac{\mathtt{Im}^{(2)}\left(f(x + hi_1, y + hi_2)\right)}{h^2},\end{cases} \tag{17}$$

where $\mathtt{Im}^{(2)}$ picks the mixed imaginary direction of $i_1 i_2$. One can easily tell from Eq. (17) that second-order CSFD is also subtraction-free making them as robust/accurate as the first-order case. With CSFD, we augment the DAE to allow each neuron to house a complex

or a multicomplex quantity. Therefore, the input perturbation can be passed through the network for computing its derivative values.

4.2 Differentiation under Tensor Contraction

A limitation of CSFD lies in its dependency on the perturbation. If the function takes m input variables e.g., an m-dimension vector, CSFD needs to evaluate the function for m times in order to compute its first-order derivative. In our case, the function is shaped as a DAE. More precisely, $D(\mathbf{q})$ corresponds to the decoding part of the network (Fig. 3). We need to take a forward pass of the decoding network as one function evaluation. The total number of network forwards goes up exponentially with respect to the order of differentiation. Therefore, computing $\mathbb{S}$ in Eq. (12) requires n_q^3 network forwards per Newton iteration, which is further scaled by the complexity D. This is too expensive for real-time simulation even on GPU.

An important contribution of this work is to efficiently enable high-order differentiability of DAE (or other deep networks) while eliminating excessive network perturbations. Our method is based on two following key observations:

- In CSFD, the imaginary parts can be somehow understood as the differential change induced by the perturbation. Under a straight usage, CSFD is analogous to forward automatic differentiation (AD) [Guenter 2007], but with much better generalization to higher orders. The potential of CSFD is maximized if the function has a high-dimension output so that one function evaluation gives you more information of the differentiation. Conversely, the BP procedure of a neural net is essentially a reverse AD [Baydin et al. 2017] – its efficiency is optimal when the input of a network is in high dimension. This is exactly the case in neural net training, where we have a large number of network parameters as the function input. It is clear that CSFD and BP nicely complement each other so that we can choose the direction of network propagation accordingly.

- While high-order differentiation produces high-order tensors, those tensors are rarely needed in its original form. In most cases, they are to be "reduced" by tensor contractions with other tensors left and right to them. Those reduction operations allow us to apply the perturbation *collectively*, not at an individual variable but in the form of vector or tensor.

4.3 Right Contraction via Directional Derivative

We now elaborate our method first with a toy example. Consider $f : \mathbb{R}^m \to \mathbb{R}$. Computing its Hessian ($\mathbf{H} = \nabla^2 f$) will need m^2 perturbations with second CSFD (Eq. (17)). However, if $\nabla^2 f$ is also contracted with a right vector $\mathbf{a}$, $\mathbf{Ha}$ can actually be evaluated much more efficiently as:

$$
\begin{aligned}
[\mathbf{H}(\mathbf{x})\mathbf{a}]_k &= \sum_{l=0}^{m-1} \lim_{h\to 0} \frac{[\nabla f(\mathbf{x}+h\mathbf{e}_l) - \nabla f(\mathbf{x})]_k}{h} \cdot [\mathbf{a}]_l, \\
\Rightarrow \quad [\mathbf{H}(\mathbf{x})\mathbf{a}]_k &= \sum_{l=0}^{m-1} \lim_{h\to 0} \frac{[\nabla f(\mathbf{x}+[\mathbf{a}]_l h\mathbf{e}_l) - \nabla f(\mathbf{x})]_k}{h}, \\
\Rightarrow \quad \mathbf{Ha} &= \lim_{h\to 0} \frac{\nabla f(\mathbf{w}+h\mathbf{a}) - \nabla f(\mathbf{x})}{h} \approx \frac{\mathtt{Im}(\nabla f(\mathbf{x}+hi\cdot\mathbf{x}))}{h}.
\end{aligned}
$$

Here, $[\cdot]_k$ gives k-th element of vector, and $\mathbf{e}$ is the canonical bases. In the second line of the derivation, we substitute h with $[\mathbf{a}]_l h$ to cancel the multiplication of $[\mathbf{a}]_l$. One may now recognize that $\mathbf{Ha}$ is essentially the *directional derivative* of $\nabla_{\mathbf{a}} f$.

This finding is not new and has been used in Jacobian-free solvers [Knoll and Keyes 2004]. However, we note that this strategy can also be generalized for high-order cases. As shown in Fig. 4, one differentiation operation lifts the order of the resulting tensor by one. A right contraction of the tensor undoes this expansion so that the perturbation can be applied together. Now let us advert to the forth tensor $\mathbb{S}$ in Eq. (12). Its exact form is of less interest to us. Instead, we would like to compute the matrix after two contractions with $\mathbf{q} - \bar{\mathbf{q}}$. To this end, we apply a collective third-order multicomplex perturbations to the decoding DAE for n_q times. The j-th perturbation computes the j-th column of the resulting matrix. This perturbation is applied along the first imaginary direction i_1 at the j-th element of the DAE input $\mathbf{q}$. The perturbations in i_2 and i_3 are scaled by the corresponding elements in $\mathbf{q} - \bar{\mathbf{q}}$. Putting together, the j-th element, which is a third-order multicomplex quantity, of the CSFD input is:

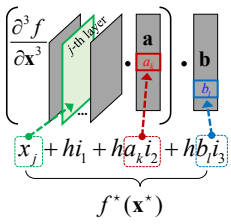

Fig. 4. Right contraction can be dealt with by applying CSFD perturbation collectively.

$$
[\mathbf{q}^\star]_j = [\mathbf{q}]_j + hi_1 + h[\mathbf{q} - \bar{\mathbf{q}}]_j i_2 + h[\mathbf{q} - \bar{\mathbf{q}}]_j i_3. \tag{18}
$$

After the forward pass, we extract the component at $i_1 i_2 i_3$ direction, and divide it by h^3.

4.4 Left Contraction via Complex-step Backpropagation

In the simulation, there are several computations involving contraction between a left vector and a differentiation tensor such as all the $\mathcal{H}^\top$ terms in Eq. (11). In those cases, the contraction occurs at the dimension which is not expanded by the differentiation. Hence, the strategy outlined in § 4.3 does not apply. Consider evaluating $\mathbf{a} \cdot \mathcal{H}$ (i.e., $\mathcal{H}^\top \mathbf{a}$). We carry out our computation with an auxiliary function $g(\mathbf{q}) = \mathbf{a} \cdot D(\mathbf{q}) \in \mathbb{R}$. As D is embodied as a neural network, this auxiliary function can also be viewed as appending an FC layer at the end of the net reducing its N-dimension output to a single scalar (like the loss function). Because $\mathbf{a}$ is independent on D, $\partial^k g / \partial \mathbf{q}^k = \mathbf{a} \cdot \partial^k D / \partial \mathbf{q}^k$. Hence, $\mathbf{a} \cdot \mathcal{H}$ can be computed as the Hessian of $g(\mathbf{q})$. Here, the reader may be reminded that $\mathcal{H}$ is a function of $\mathbf{q}$, and it is the second derivative of d. A standard second CSFD will need $n_p(n_p+1)/2$ perturbations knowing $\partial^2 g / \partial \mathbf{q}^2$ is symmetric. We show that is computation can be further reduced to $\mathbf{O}(n_q)$.

As mentioned, CSFD is most suited for differentiating functions with a high-dimension output – $g(\mathbf{q})$ is not such a function, which outputs a single scalar. Its derivative could be more efficiently computed by reverse AD or BP. As a first-order routine however, BP only computes the gradient function $\partial g / \partial \mathbf{q}$. To this end, we inject CSFD into the BP procedure treating BP as a generic function and enabling complex arithmetic along the BP computation to perturb the gradient of g. It starts with a complex-perturbed forward pass of the network g by adding the perturbation at one element (say the

j-th element) of the network input as: $[\mathbf{q}^*]_j = [\mathbf{q}]_j + hi$. The feedforward of the net delivers this complex perturbation to all the neurons $[\mathbf{q}]_j$ influences. BP then ensues. During BP, all the computations are complex-based. If a neuron receives an imaginary component in the forward pass, this imaginary component participates in BP and passes complex-value feedback signals to its previous layer. After BP, all the signals at the input layer are divided by h yielding one column of $\mathbf{a} \cdot \mathcal{H}$.

Fig. 5 illustrates this process with a simple net: two neurons (x and y) multiply first, and the result (z) is squared to generate the output (w). Suppose $x = 2$ and $y = 3$, and we want to compute the second derivative of the network output with respect to x. The perturbation h is applied to x so that $x = 2+hi$, and all sequential neurons become complex-value. After the forward pass, BP invokes. Everything remains the same as the regular BP except the computation is in complex. For instance, $\partial w/\partial z = 2z$; as z holds a complex value, $\partial w/\partial z = 12 + 6hi$ is also complex. Finally, after BP completes. The real part of x gives the value of the first-order derivative – the same as the original BP algorithm, and the imaginary part of x after being divided by the input perturbation h is the second derivative. Along this procedure, we follow the strategy in [Luo et al. 2019] to avoid unneeded complex computations. For instance in Fig. 5, high-order terms of h is discarded in w.

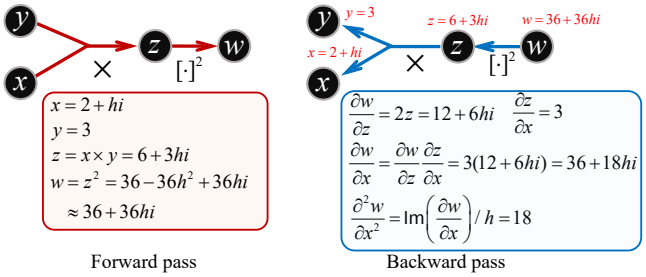

Fig. 5. By augmenting BP with CSFD, we can efficiently evaluate high-order differentiation of a deep net, followed by a left-side tensor contraction.

Thanks to CSFD, all the tensor-related computations can now be completed with n_q network passes, either forward passes with CSFD or backward passes with CSFD-enabled BP. Those n_q network passes can be executed in parallel on GPU as one single mini-batch. The remaining performance bottleneck is the subspace integration of reduced force and elastic Hessian. This computation is usually handled with the Cubature method [An et al. 2008]. In the next section, we discuss how we replace the classic Cubature sampling with a neural network based one to allow a pose-dependant subspace integration.

5 NEURAL CUBATURE SAMPLING AND WEIGHTING

In model reduction, it is expected that all the computations are carried out in the polynomial time of the reduced order $n_p + n_q$. For Saint Venant-Kirchhoff (StVK) material model under linear reduction, it is possible to pre-compute the polynomial coefficients for reduced force and Hessian [Barbič and James 2005] at the cost of $\mathbf{O}(n_p^4)$. Unfortunately, other material models do not share this

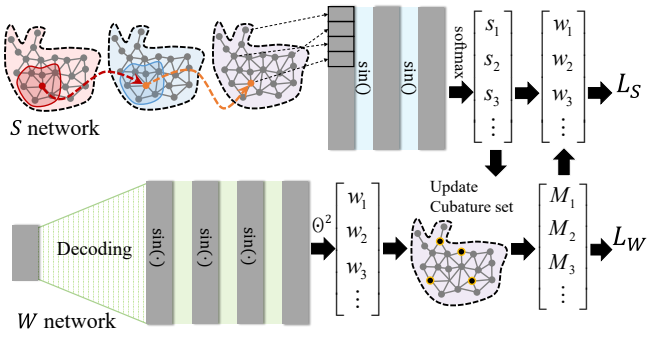

Fig. 6. Neural Cubature alternates between two neural networks: S and W. S net is a GCN and selects Cubature elements with highest scores. W net predicts the weight of each Cubature element. We add a square activation to ensure the non-negativeness of the output weight value. This information is then passed back to S for next-round selection.

convenience. A practical solution is the so-called Cubature sampling [An et al. 2008]. Cubature selects a subset of key elements (i.e., Cubature elements) such that the reduced force and reduced Hessian can be integrated only at Cubature elements with a designated non-negative weight. Cubature has been proven effective for linear model reduction. However, its naïve deployment for nonlinear reduction is questionable: as the tangent space varies in nonlinear cases, why should we stick with invariant Cubature weights?

Our neural Cubature consists of two networks as shown in Fig. 6. The first net is in charge of selecting new Cubature elements, and the second net is responsible for predicting Cubature weights. Specifically, the first neural network outputs a "score" for each element, and we can add *multiple* elements to the Cubature set based on element's score. After updating the Cubature set $\mathcal{C}$, the second network outputs the weights of all the Cubature elements based on the input $\mathbf{r}$. Neural Cubature training alternates between those two networks. After the training, only weight prediction net participates in the simulation i.e., given a generalized coordinate $\mathbf{r}$, the neural net outputs its weights coefficients at the simulation run time, which are then used for the subspace integration.

5.1 Cubature Selection with a GCN

We use S to denote the first neural network for Cubature element selection. S is a graph convolutional network (GCN) [Wu et al. 2020], which naturally inherits the topology of the input 3D model. The input of S is the fullspace displacement $\mathbf{u}$ followed by two graph convolution layers. Each convolution layer produces eight channels. After that, another two FC layers are applied. S outputs the probability s_e for each element e on the mesh, which is concatenated into a global probability or score vector $\mathbf{s}$. The graph convolution operation can be written as:

$$h_i^{(l+1)} = \sin\left(\sum_{j \in \mathcal{N}_i} \frac{1}{c_{ij}} h_j^{(l)} \gamma 1^{(l)}\right), \tag{19}$$

where $h_i^{(l)}$ represents the i-th vertex in the l-th neural network layer. $\gamma^{(l)}$ is the trainable parameter, and $\mathcal{N}$ denotes the one-ring

neighborhood of i on the mesh. Similar to DAE, we use $\sin(\cdot)$ for intermediate nonlinear activations. $c_{ij} = \sqrt{d_i \cdot d_j}$ is the normalization constant of edge $\langle i, j \rangle$, where d_i is the degree of vertex i. At the last hidden layer, we use the softmax activation [Goodfellow et al. 2016], which assigns each element a probability score.

S is trained in the *residual space*. This scheme is inspired by the original Cubature algorithm. At the beginning, the set of Cubature elements is empty: $\mathcal{C} = \emptyset$, and the original training set consists of pose-force pairs. With some elements being selected, $\mathcal{C} \neq \emptyset$, we compute the remaining reduced force for each training data with current Cubature integration:

$$\bar{\mathbf{f}}(\mathbf{r}) = \widetilde{\mathbf{f}}_{int}(\mathbf{r}) - \sum M_e(\mathcal{C}, \mathbf{w}) \widetilde{\mathbf{f}}_e(\mathbf{r}). \tag{20}$$

Here, $\widetilde{\mathbf{f}}_{int}(\mathbf{r})$ is the reduced internal force projected in the column space of $\widetilde{\mathbf{J}}(\mathbf{r})$. The summation iterates all the elements on the mesh. $M_e(\mathcal{C}, \mathbf{w})$ is a *mask* function that removes non-Cubature weights from an input weight vector $\mathbf{w}$. In other words, $M_e(\mathcal{C}, \mathbf{w}) = [\mathbf{w}]_e$ if element $e \in \mathcal{C}$ or 0 otherwise. Note that the dimensionality of $\mathbf{w}$ corresponds to the total number of elements on the model, and it is the output from the current weight prediction net. $\widetilde{\mathbf{f}}_e$ is the reduced force at the element e. Instead of adding one Cubature element each time, neural Cubature allows us to select multiple elements. After S outputs scores $\mathbf{s}$, we can pick K non-Cubature elements with highest scores, and update $\mathcal{C}$ accordingly. We have tested $K = 5$, $K = 10$, and $K = 20$ and did not find much difference between them.

5.2 Weight Prediction

The weight prediction network W takes a generalized coordinate $\mathbf{r}$ as well as the current Cubature set $\mathcal{C}$ as input, and outputs weight coefficients for all the elements $\mathbf{w}$. Specifically, $\mathbf{r}$ is first spanned to $\mathbf{u}$ with our decoder net. Network parameters at this part are fixed and do not participate in the training. Four additional FC layers with $\sin(\cdot)$ activations are followed. W is not a graph network, as we believe the geometry and topology information of the model is already captured in S. Training the weight should be pure algebraic, and several nonlinearly activated FC layers work for this purpose well. Because W is the part of the simulation (we need to obtain $\mathbf{w}$ at each time step), we also want to make sure it is light-weight and runs feedforward efficiently. Therefore, the structure of W is plain and straightforward. Finally, the weight coefficients of Cubature elements should be non-negative in order to prevent extrapolation and overfitting. To this end, we put a square operation at the last layer to enforce the non-negative constraint.

The loss functions of both S and W resemble each other a lot:

$$\begin{aligned} L_S &= \left\| \bar{\mathbf{f}}(\mathbf{r}) - \sum M_e(\mathcal{C}, \mathbf{w}) \widetilde{\mathbf{f}}_e(\mathbf{r}) \right\|, \\ L_W &= \left\| \bar{\mathbf{f}}(\mathbf{r}) - \sum M_e(\mathcal{C}, \mathbf{w}) \widetilde{\mathbf{f}}_e(\mathbf{r}) \right\|. \end{aligned} \tag{21}$$

In practice, neural Cubature kicks off by initializing $\mathcal{C}$ as few Voronoi samples of the input model, which are passed to W to start the alternating. After $\mathbf{w}$ is predicted, we feed this information to S (i.e., updating the Cubature residual), which in turn, updates the Cubature set $\mathcal{C}$. Our neural Cubature is more efficient and accurate than conventional Cubature methods. Because neural Cubature picks multiple elements each time, we can also quickly build a bigger Cubature set $\mathcal{C}$.

6 EXPERIMENTAL RESULTS

We have implemented our framework on a desktop computer with an `intel i7 9700` CPU and an `nVidia 2080` GPU. The simulation part is mostly on CPU but we move all the matrix-matrix and matrix-vector computations to GPU with `cuBLAS` [Nvidia 2008]. The simulation is implemented with `C++`, and some linear algebra computations are based on `Eigen` library [Guennebaud et al. 2010]. Network training is initially carried out using `PyTorch` [Paszke et al. 2019]. After we have all the network parameters, we port the resulting neural network to `CUDA`. Network BP for computing tensor contraction is also implemented with `cuBLAS`.

6.1 Training Poses Generation

We generate training poses by running a scripted simulation. At the training stage, given a random surface vertex on the model, we select its nearby vertices within a given radius, and apply a random force to them (Fig. 7). All the simulation poses along this dynamic procedure are recorded as training data.

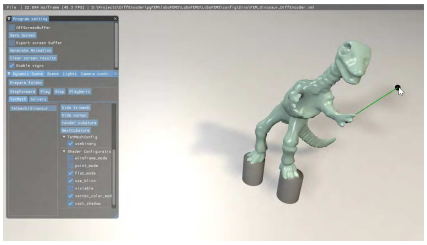

Fig. 7. We generate training data by applying scripted random forces to the model.

In linear model reduction, it is common to directly sample training data within the modal space e.g., see [Von-Tycowicz et al. 2015]. We found that this strategy is not valid for nonlinear model reduction. Here, we would like to clarify two confusing concepts: pose and basis. In linear model reduction, we care more about the basis, whose most important attribute is the direction, and its magnitude matters little. This is not the case for nonlinear reduction, where we essentially learn the underlying deformation manifold. An effective training will need samples on this manifold i.e., poses, without unnecessary scaling. Therefore, training data should be generated via real simulation.

Training poses ought not be weighted equally. In general, we prefer to better fit poses closer to the rest shape. A slightly higher fitting error may be acceptable for poses under large deformations. This is also the motivation in the linear model reduction of scaling basis vectors by their vibration frequencies [Barbič and James 2005; Von-Tycowicz et al. 2015]. However, nonlinear eigenvalues of deformable poses are difficult to be estimated. We found a good metric is the elastic energy of a given pose, which nonlinearly measures how far a deformation is away from the rest configuration. As a result, we weight the loss value of each pose by the inverse of its elastic energy. As discussed, our method also builds a linear subspace (i.e., $\mathcal{S}_p$) via PCA out of the training poses. If the training data set is too big, computing a full PCA is time-consuming. We find that a good walk-around is to randomly pick poses with smallest elastic

energy to form a more compact training set for PCA, and leave DAE to extract nonlinear information out of the residual pose space.

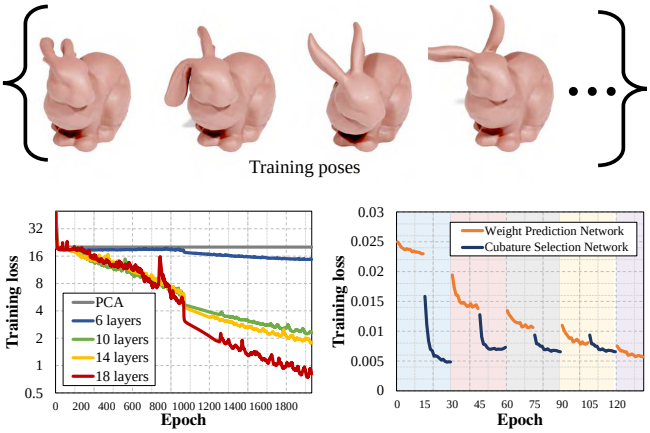

Fig. 8. Network curves for training the bunny model. Neural Cubature is trained by alternating S and W nets, and we use the parameters from the previous alternation. Adding more layer helps reduce the training error effectively.

6.2 Network Training

We use `PyTorch` and Adam for all our network training. For training DAE, we start with an initial learning rate of 0.001. After 300 epochs, we shrink the learning rate by 20%, and another 20% after 3,000 epochs. Normally, a training set includes 20,000 poses for a model. When training the neural Cubature networks S and W, we stick with the learning rate of 0.001. In each alternation, we run 15 epochs for both S and W. Depending on how many Cubature elements we want to pick, the neural Cubature training could take several thousand epochs. The total network training time is less than expected, which takes ten to twenty minutes. Generating the training poses is the most expensive part. It often needs a couple of hours. A typical training curve is reported in Fig. 8, which is for the bunny model. We note that the expressivity of the DAE improves with increased depth. This can be observed from Fig. 8: if the DAE is shallow e.g., fours layers, its performance is only marginally better than PCA; but with a deeper DAE, the error decreases sharply.

6.3 Comparison I: Our Method vs. Linear Model Reduction

First, we report a comprehensive comparative experiment between our DAE-based nonlinear reduction and other commonly seen linear reduction methods including: PCA, physical modal derivative (PMD) [Barbič and James 2005], and geometric modal derivative (GMD) [Von-Tycowicz et al. 2015]. Both physical and geometric modal derivatives are based on linear modal analysis (LMA) [Pentland and Williams 1989]. PMD is computed via solving a set of static equilibria around the rest shape, while GMD constructs the subspace matrix by spanning each LMA basis to nine tangent directions corresponding to its local linear transformation. In this experiment, we first compute 50 LMA basis vectors. Based on them, we compute $50 \times (50+1)/2 = 1,275$ PMD bases and $50 \times 9 = 450$ GMD bases. Finally, we apply mass-PCA as described in [Barbič and James 2005] to extract the subspace matrix for the linear reduction. We report the results with a dinosaur model because of its concave and non-trivial geometry.

Fig. 9. The ground truth poses of "bending" (left) and "opening arm" (right) of the dinosaur model.

In the first set of comparison, as shown in Fig. 10 (left), we fix the feet of the model and bend the dinosaur backwards. We compare the final poses of different reduction methods under different subspace sizes: 10, 15, and 20. Our method adopts a mixed linear and nonlinear subspaces superposition, the dimensionality of each subspace is set as $n_p = 5$, $n_q = 5$; $n_p = 10$, $n_q = 5$; and $n_p = 10$, $n_q = 10$. The ground truth shape is given in Fig. 9 (left).

In this experiment, we can see a clear advantage of our method over PCA-based linear reduction. We think the reason is straightforward, DAE is known to be more expressive than conventional PCA especially for nonlinear data sets. In addition, we find that PMD also gives very good results while GMD does not perform well. We assume this is because you need to fully incorporate all 450 geometric derivative modes in GMD to first-order approximate the derivative of LMA modes reasonably well. PMD is optimal for low-frequency deformations like this dinosaur bending. Indeed, PMD is exactly designed to capture such deformations, while our method is based on a data set generated randomly. From this perspective, it is actually encouraging to see our method yields comparable results in PMD's "home field". Another common trend for all the reduction methods is that the deformation improves with increased subspace dimensions.

To further verify our hypothesis, we generate another set of training poses (500 poses), where we only add random forces at the hands of the dinosaur. This type of deformation is local and high-frequency, which are less friendly for PMD as the bending pose. In the test, we ask the dinosaur to open its arm by applying forces to its hands outwards. All the other settings remain unchanged. As shown in Fig. 10 (right), the difference between our method and PMD becomes more obvious in this experiment. The ground truth result is the right snapshot of Fig. 9. Interestingly, when we narrow our training sampling at the hands, the performance of PCA also gets much better. We can see from the figure that PCA is very close to our method. This is because local deformation does not necessarily suggest higher nonlinearity. In the "opening arm" test, we only generate 500 training poses, which can be fairly well captured by PCA. The advantage of nonlinear reduction is more observable when the subspace size is further condensed (e.g., when $n = 10$).

6.4 Comparison II: Our Method vs. Latent Space Dynamics

We are not the first to leverage DAE to perform nonlinear reduction. Latent space dynamics (LSD) [Fulton et al. 2019] is closely relevant to our method. Both our method and LSD share the same high-level

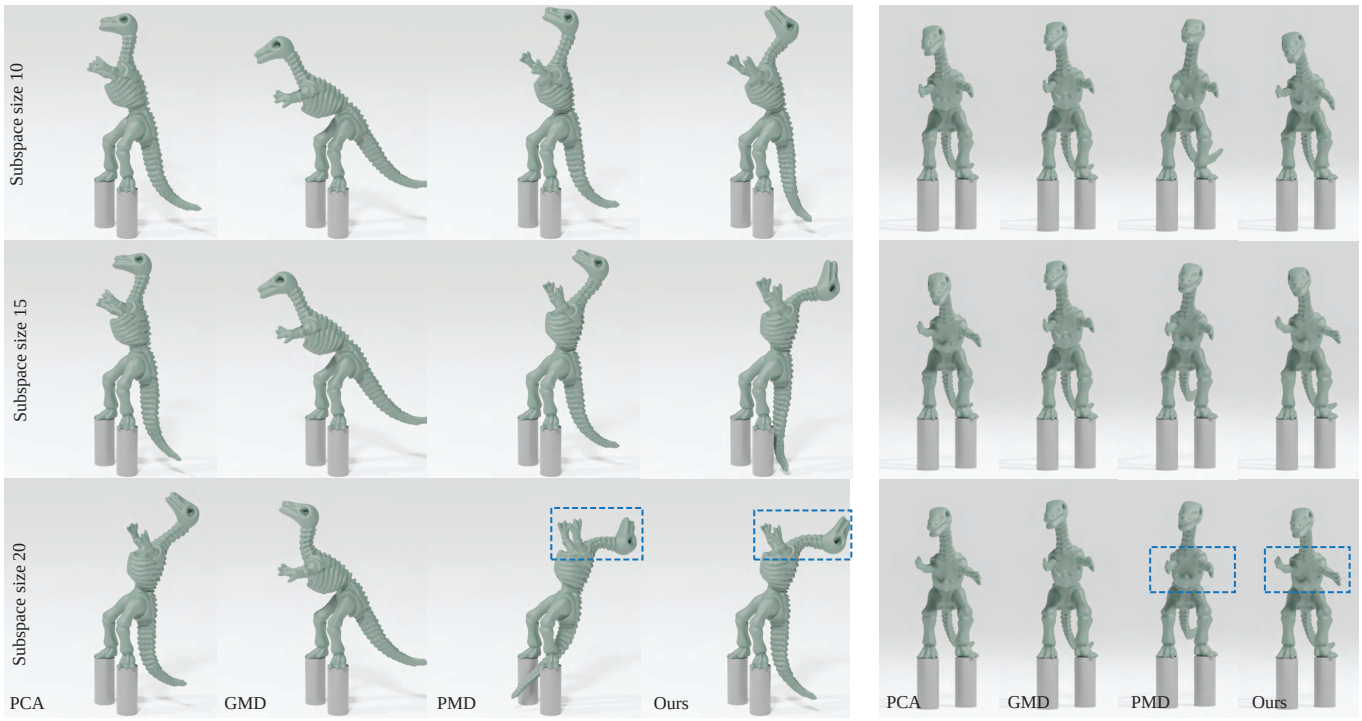

Fig. 10. We compare simulation results of the dinosaur model using various model reduction methods: PCA, physical modal derivative (PMD), geometric modal derivative (GMD), and our method. The ground truth shapes are given in Fig. 9. On the left, we globally bend the dinosaur, and on the right, we try to apply local forces at its hands.

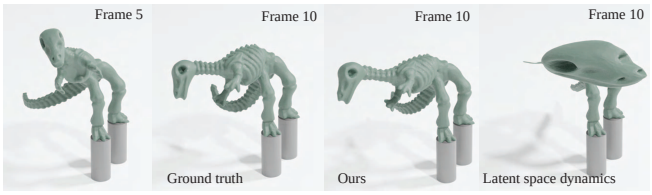

Fig. 11. Latent space dynamics [Fulton et al. 2019] uses a trimmed formulation to avoid the evaluation of high-order derivative of DAE. This simplification leads to serious artifacts when the model moves under a high velocity.

motivation of nonlinear subspace simulation, and both choose to use autoencoder as the machinery of the reduction in elastic simulation. Therefore, we consider LSD our major competitor. There are several key differences between our method and LSD. The most important one lies in the fact that the lack of differentiability in LSD needs a simplified formulation that ignores the fictitious force $\mathbf{f}_{fict}$ (Eq. (8)). This could lead to significant error during the simulation when the model undergoes a high-velocity motion.

Fig. 11 reports snapshots of this issue. In this test, we drag the head of the dinosaur to left with an abrupt force. The artifact does not appear serious at first few frames. However, once the accumulated error reaches a certain level, the simulation diverges and cannot be recovered even we slow down the animation later. In order to have a fair comparison, we run our simulation fully in $\mathcal{S}_q$ without building the PCA space $\mathcal{S}_p$, and we do not use Cubature sampling for subspace force integration to avoid other potential error sources.

As most dynamic simulation problems are prescribed by Newton's law of motion, being able to evaluate high-order derivatives of nonlinear model reduction is a must for a successful deployment of this technique. This turns out be the key contribution of our method. In LSD, there are many smart strategies used to remedy the risk induced by the missed $\mathbf{f}_{fict}$ such as pre PCA filtering in the DAE network etc. They are all compatible with our method, but the $\mathbf{f}_{fict}$ issue does not even exist in our framework.

	10	20	50	100	200
Neural Cubature [10]	91.1%	62.9%	39.1%	23.8%	16.3%
Neural Cubature [5]	87.1%	60.3%	37.1%	22.3%	15.7%
Greedy Cubature	88.4%	66.6%	47.1%	31.2%	19.2%

Table 1. Cubature sampling error using neural Cubature and classic Cubature method. This experiment is performed on an Armadillo model with 38K elements. Neural Cubature [10] means we add 10 elements to the Cubature set $\mathcal{C}$ based on each S network prediction. Neural Cubature [5] adds 5 elements each time. Neural Cubature outperforms classic Cubature method by a substantial margin: in general neural Cubature yields $30-40\%$ less error than classic greedy Cubature method.

6.5 Comparison III: Neural Cubature vs. Classic Cubature

In the next experiment, we would like to investigate the difference between our neural Cubature and the classic Cubature method. We

first compare the fitting error of 10, 20, 50, 100, and 200 Cubature elements. The results are reported in Tab. 1.

We can see from listed error percentages that neural Cubature outperforms classic Cubature method [An et al. 2008] in the context of nonlinear reduction. The selection network S of neural Cubature uses a GCN, which captures the geometry and topology information of the input model, while classic Cubature method is solely algebraic. Another advantage of neural Cubature is its efficiency. Neural Cubature allows us to choose multiple Cubature elements each time when S net predicts a score vector. We find that the Cubature training error is not sensitive to how many new Cubature elements we add to $\mathcal{C}$ every time, as long as this is a reasonable number i.e., in dozens – picking five elements has a higher accuracy than picking ten elements, but both are better than greedy Cubature (Tab. 1).

In addition, we can reuse the network parameters from the previous alternation to warm start the training for a faster convergence (e.g., see Fig. 8). With neural Cubature, we can conveniently build a bigger $\mathcal{C}$ set. Under CUDA-assisted subspace integration, the neural Cubature sampling error can be effectively suppressed. This is hardly possible with classic Cubature method as we need to solve a non-negative least square problem with increasing size. Building $\mathcal{C}$ of few hundred elements would be very expensive. Fig. 12 shows a concrete experiment of simulating an Armadillo using greedy and neural Cubature strategies with 140 Cubature elements. We fix Armadillo's hand and drag its leg downwards. Our neural Cubature with varying weights simulates this animation robustly while greedy Cubature fails ($n_p = 10$ and $n_q = 10$). In this experiment, one needs to increase greedy Cubature samples to over 250 to reduce the error in the integration, which takes a few more training hours.

Fig. 12. We drag the left leg of the Armadillo. Neural Cubature with 140 elements runs the simulation robustly. We also visualize the weight value at each Cubature element. Higher weighted elements are brighter. Greedy Cubature fails in this simulation.

6.6 Comparison IV: CSFD vs. Finite Difference

We have briefly discussed in § 4 that finite difference is not numerically robust even for first-order cases. Its applicability in nonlinear reduction is unlikely possible. To verify this, we implement second- and third-order differentiation by recursively applying center finite difference. The simulation does not converge no matter how we tweak the perturbation size h: $h = 1E-3$, $h = 1E-5$, $h = 1E-7$, $h = 1E-9$. In fact, the simulation crashes almost immediately when finite difference is used. We believe the increased depth of the neural net imposes more challenges for finite difference to work probably. However, CSFD is robust and accurate even for high-order differentiations.

6.7 Implementation Details

We first use PyTorch to test and train our neural networks (DAE, S and W). After the training is complete, we re-implement the net with CUDA, which is directly implanted in our simulation framework. Our CUDA port is CSFD-capable, i.e., the forward and backward pass of the network also takes multi-/complex values. This could be done by overloading the real operators with their complex or multicomplex counterparts.

Alternatively, we choose to use the *Cauchy-Riemann* (CR) formulation [Ahlfors 1973; Luo et al. 2019] to achieve (multi-)complex perturbations without overloading the complex arithmetic. CR equation represents a multicomplex number in the form of a real matrix. Suppose $z^1 = z_0^0 + z_1^0 i$, its CR form is a 2×2 matrix:

$$z^1 = z_0^0 + z_1^0 i = \begin{bmatrix} z_0^0 & -z_1^0 \\ z_1^0 & z_0^0 \end{bmatrix}, \text{ where } z^1 \in \mathbb{C}^1 \text{ and } z_0^0, z_1^0 \in \mathbb{C}^0 = \mathbb{R}.$$

Here, we use the superscript $(\cdot)^n$ to denote the order of a multicomplex number. The CR matrix of z^n can be constructed recursively using the CR matrices of z_0^{n-1} and z_1^{n-1} as:

$$z^n = z_0^{n-1} + z_1^{n-1} i_n \in \mathbb{C}^n = \begin{bmatrix} z_0^{n-1} & -z_1^{n-1} \\ z_1^{n-1} & z_0^{n-1} \end{bmatrix}. \tag{22}$$

Each of the 2×2 blocks in Eq. (22) is a $(n-1)$-order multicomplex number, which can be further expanded with $(n-2)$-order multicomplex numbers and so on. Eventually, the CR form of z^n becomes a $2^n \times 2^n$ real matrix.

With CR formula, we organize each network layer into a real layer and an imaginary layer (or multiple multicomplex layers) other than generalizing each neuron to be a complex or multiple quantity. All the computations are now in real, and we implement the forward pass of the net for FC layers with cuBLAS. The CR matrix multiplication is carried out block-wisely, so that we do not generate redundant multiplications corresponding to the off diagonal blocks.

The activation is on the other hand, directly implemented by launching CUDA threads. Fortunately, we do not have many different types of activations. Only the periodic activation function $\sin(\cdot)$ is used. To this end, we just implement its naïve expression up to the third order to maximize the performance on CUDA without recursive variable initialization. While the expression looks verbose (e.g., in Appendix A and the supplementary document), computing the high-order derivative of activation function only takes a small fraction of the network forwards. The major computing efforts remain at FC forward and backward passes.

6.8 Extensions and More Results

Our DAE-based nonlinearly reduced simulation algorithm can be extended and integrated into other simulation frameworks at ease. For instance, we can use the generalized Newton-Euler equation to couple local deformation and rigid body dynamics [Kim and Delaney 2013; Shabana 2003]. The training poses need to be generated with rigid body motion removed as well in this case. Fig. 13 reports a real-time simulation of a falling bunny on wooden stairs. We use implicit penalty force to resolve the collision and self-collision.

Fig. 13. Falling bunny. We use generalized Newton-Euler equation to couple DAE-based model reduction with large rigid body motion to simulate free-floating objects. The subspace configuration of the bunny is $n_p = 30$ and $n_q = 10$.

	# Ele.	# Tri.	$n_p + n_q$	$\|\mathcal{C}\|$	# D.	FPS
Dinosaur	18K	9K	30 + 10	100	–	44 (31×)
Armadillo	40K	20K	30 + 10	140	–	30 (35×)
Bunny	16K	8K	30 + 10	100	–	45 (30×)
Cactus	233K	139K	10 + 6	6,600	165	2.5 (56×)
Puffer ball	625K	120K	10 + 5	11,400	321	1.4 (36×)

Table 2. Time performance of our nonlinear subspace simulator. # Ele. and # Tri. are the total numbers of elements and surface triangles on the model; $n_p + b_q$ reports the composition of our subspace configuration (per domain); $|\mathcal{C}|$ is the number of Cubature elements used; # D. is the total number of domains on the model; FPS is the simulation frame per second (and speedups compared with single-core full simulation).

Fig. 14. Dropping an Armadillo into cactus. We use deformation substructuring [Barbič and Zhao 2011] method to build multi-level subspaces at the cactus. Each local domain has a compact subspace of $n_p = 10$ and $n_q = 6$.

For geometric-complex models, the advantage of nonlinear reduction could be further amplified with the domain decomposition method [Barbič and Zhao 2011; Wu et al. 2015; Yang et al. 2013] instead of naïvely increasing the global subspace size. To this end, we also couple our DAE-based nonlinear reduction with deformation substructuring [Barbič and Zhao 2011], which delivers more interesting animations with DAE-enriched local details. Two examples are reported in Figs. 1 and 14. The puffer ball is an ideal vehicle to deliver the advantage of this generalization. It has 320 elastic strings with the same geometry. Therefore, the network training (for both DAE and neural Cubature) of one string can be re-used for all other strings. Thanks to its simple and symmetric geometry, depth of DAE can also be cut to 8. The cactus example shown in Fig. 14 is another representative case: the tree-like structure of the cactus allows an effective hierarchical deployment of nonlinear subspaces. Here, we have 165 domains on the cactus, and each domain has a subspace of $n_p = 10$ and $n_q = 6$. Lastly, the simulation time performance is summarized in Tab. 2.

7 CONCLUSION AND LIMITATION

In this paper, we present a framework combining classic reduced deformable simulation with deep learning empowered data-driven approaches. We advance state-of-the-art reduction methods by plugging a deep autoencoder into the simulation pipeline. While some existing work has attempted this idea before, we are the first to address the high-order differentiability of the deep neural net in order to accurately project nonlinear dynamics of deformable solids into the tangent space of the deformation manifold. This is made possible by carefully re-engineering complex-step finite difference in the context of deep learning and complementing CSFD with reverse AD. With a CSFD-augmented BP and CSFD directional derivatives, we can evaluate the high-order derivatives of a deep net only with $\mathbf{O}(n)$ network passes. Based on this, we also propose a neural Cubature scheme that allows a more efficient Cubature sampling and more accurate weighting. Without ignoring inertia forces induced by the time-varying tangent projection, we are able to simulate deformable objects with nonlinear model reduction in real time robustly. We believe CSFD-enabled differentiability paves the way to an in-depth integration of deep neural network and physics-based simulation, which could inspire many follow-up research efforts.

There are also several limitations of our framework. First, the visual improvement of our nonlinear reduction method over existing linear reduction method is not "wow". After all, we are manipulating a reduced simulation with only dozens of DOFs. We believe combing neural network and other data-driven approaches used in graphics could potentially improve this issue. For instance, if the further deformation types are somehow known, we could use DAE to build

a more specific nonlinear subspace as in [Harmon and Zorin 2013]. Building hierarchical DAE is also a promising solution. It may be possible to train a neural network to select multiple pre-trained DAEs to locally expand the tangent space. Model reduction is a powerful tool not only for deformable object simulation. To this end, we will further investigate how to use nonlinear reduction to improve other simulation problems like fluid, cloth, and sound synthesis.

REFERENCES

Lars V Ahlfors. 1973. Complex Analysis. 1979.

Steven S An, Theodore Kim, and Doug L James. 2008. Optimizing cubature for efficient integration of subspace deformations. *ACM transactions on graphics (TOG)* 27, 5 (2008), 1–10.

Johannes Ballé, Valero Laparra, and Eero P Simoncelli. 2016. End-to-end optimized image compression. *arXiv preprint arXiv:1611.01704* (2016).

Jernej Barbič, Marco da Silva, and Jovan Popović. 2009. Deformable object animation using reduced optimal control. In *ACM SIGGRAPH 2009 papers*. 1–9.

Jernej Barbič, Funshing Sin, and Eitan Grinspun. 2012. Interactive editing of deformable simulations. *ACM Transactions on Graphics (TOG)* 31, 4 (2012), 1–8.

Jernej Barbič and Yili Zhao. 2011. Real-time large-deformation substructuring. *ACM transactions on graphics (TOG)* 30, 4 (2011), 1–8.

Jernej Barbič and Doug L James. 2005. Real-time subspace integration for St. Venant-Kirchhoff deformable models. In *ACM Trans. Graph. (TOG)*, Vol. 24. ACM, 982–990.

Atılım Günes Baydin, Barak A Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. 2017. Automatic differentiation in machine learning: a survey. *The Journal of Machine Learning Research* 18, 1 (2017), 5595–5637.

Hervé Bourlard and Yves Kamp. 1988. Auto-association by multilayer perceptrons and singular value decomposition. *Biological cybernetics* 59, 4-5 (1988), 291–294.

Alain J Brizard. 2014. *Introduction To Lagrangian Mechanics, An.* World Scientific Publishing Company.

H Martin Bücker, George Corliss, Paul Hovland, Uwe Naumann, and Boyana Norris. 2006. *Automatic differentiation: applications, theory, and implementations*. Vol. 50. Springer Science & Business Media.

Steve Capell, Seth Green, Brian Curless, Tom Duchamp, and Zoran Popović. 2002. Interactive skeleton-driven dynamic deformations. In *ACM Trans. Graph. (TOG)*, Vol. 21. ACM, 586–593.

Min Gyu Choi and Hyeong-Seok Ko. 2005. Modal warping: Real-time simulation of large rotational deformation and manipulation. *IEEE Trans. on Visualization and Computer Graphics* 11, 1 (2005), 91–101.

François Faure, Benjamin Gilles, Guillaume Bousquet, and Dinesh K Pai. 2011. Sparse meshless models of complex deformable solids. In *ACM Trans. Graph. (TOG)*, Vol. 30. ACM, 73.

Marco Fratarcangeli, Valentina Tibaldo, and Fabio Pellacini. 2016. Vivace: A practical gauss-seidel method for stable soft body dynamics. *ACM Transactions on Graphics (TOG)* 35, 6 (2016), 1–9.

Lawson Fulton, Vismay Modi, David Duvenaud, David IW Levin, and Alec Jacobson. 2019. Latent-space Dynamics for Reduced Deformable Simulation. In *Computer graphics forum*, Vol. 38. Wiley Online Library, 379–391.

Benjamin Gilles, Guillaume Bousquet, Francois Faure, and Dinesh K Pai. 2011. Frame-based elastic models. *ACM Trans. Graph. (TOG)* 30, 2 (2011), 15.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. 6.2. 2.3 softmax units for multinoulli output distributions. *Deep learning* (2016), 180–184.

Gaël Guennebaud, Benoit Jacob, et al. 2010. Eigen. *URl: http://eigen. tuxfamily. org* (2010).

Brian Guenter. 2007. Efficient symbolic differentiation for graphics applications. In *ACM SIGGRAPH 2007 papers*. 108–es.

Amirhossein Habibian, Ties van Rozendaal, Jakub M Tomczak, and Taco S Cohen. 2019. Video compression with rate-distortion autoencoders. In *Proceedings of the IEEE International Conference on Computer Vision*. 7033–7042.

Fabian Hahn, Bernhard Thomaszewski, Stelian Coros, Robert W Sumner, Forrester Cole, Mark Meyer, Tony DeRose, and Markus Gross. 2014. Subspace clothing simulation using adaptive bases. *ACM Transactions on Graphics (TOG)* 33, 4 (2014), 1–9.

David Harmon and Denis Zorin. 2013. Subspace integration with local deformations. *ACM Transactions on Graphics (TOG)* 32, 4 (2013), 1–10.

Kris K Hauser, Chen Shen, and James F O'Brien. 2003. Interactive Deformation Using Modal Analysis with Constraints.. In *Graphics Interface*, Vol. 3. 16–17.

Florian Hecht, Yeon Jin Lee, Jonathan R Shewchuk, and James F O'Brien. 2012. Updated sparse cholesky factors for corotational elastodynamics. *ACM Trans. Graph. (TOG)* 31, 5 (2012), 123.

Robert Hecht-Nielsen. 1992. Theory of the backpropagation neural network. In *Neural networks for perception*. Elsevier, 65–93.

Geoffrey E Hinton and Ruslan R Salakhutdinov. 2006. Reducing the dimensionality of data with neural networks. *science* 313, 5786 (2006), 504–507.

Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, Jürgen Schmidhuber, et al. 2001. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies.

Doug L James and Christopher D Twigg. 2005. Skinning mesh animations. *ACM Transactions on Graphics (TOG)* 24, 3 (2005), 399–407.

Theodore Kim and John Delaney. 2013. Subspace fluid re-simulation. *ACM Transactions on Graphics (TOG)* 32, 4 (2013), 1–9.

Theodore Kim and Doug L James. 2009. Skipping steps in deformable simulation with online model reduction. In *ACM Trans. Graph. (TOG)*, Vol. 28. ACM, 123.

Dana A Knoll and David E Keyes. 2004. Jacobian-free Newton–Krylov methods: a survey of approaches and applications. *J. Comput. Phys.* 193, 2 (2004), 357–397.

L'ubor Ladickỳ, SoHyeon Jeong, Barbara Solenthaler, Marc Pollefeys, and Markus Gross. 2015. Data-driven fluid simulations using regression forests. *ACM Transactions on Graphics (TOG)* 34, 6 (2015), 1–9.

Lei Lan, Ran Luo, Marco Fratarcangeli, Weiwei Xu, Huamin Wang, Xiaohu Guo, Junfeng Yao, and Yin Yang. 2020. Medial Elastics: Efficient and Collision-Ready Deformation via Medial Axis Transform. *ACM Transactions on Graphics (TOG)* 39, 3 (2020), 1–17.

Gregory Lantoine, Ryan P Russell, and Thierry Dargent. 2012. Using multicomplex variables for automatic computation of high-order derivatives. *ACM Transactions on Mathematical Software (TOMS)* 38, 3 (2012), 1–21.

Tiantian Liu, Adam W. Bargteil, James F. O'Brien, and Ladislav Kavan. 2013. Fast Simulation of Mass-spring Systems. *ACM Trans. Graph. (TOG)* 32, 6 (2013), 214:1–214:7.

Tiantian Liu, Sofien Bouaziz, and Ladislav Kavan. 2017. Quasi-newton methods for real-time simulation of hyperelastic materials. *ACM Transactions on Graphics (TOG)* 36, 3 (2017), 1–16.

Ran Luo, Weiwei Xu, Tianjia Shao, Hongyi Xu, and Yin Yang. 2019. Accelerated complex-step finite difference for expedient deformable simulation. *ACM Transactions on Graphics (TOG)* 38, 6 (2019), 1–16.

Alireza Makhzani, Jonathon Shlens, Navdeep Jaitly, Ian Goodfellow, and Brendan Frey. 2015. Adversarial autoencoders. *arXiv preprint arXiv:1511.05644* (2015).

Sebastian Martin, Peter Kaufmann, Mario Botsch, Eitan Grinspun, and Markus Gross. 2010. Unified simulation of elastic rods, shells, and solids. In *ACM Trans. Graph. (TOG)*, Vol. 29. ACM, 39.

Joaquim RRA Martins, Peter Sturdza, and Juan J Alonso. 2003. The complex-step derivative approximation. *ACM Transactions on Mathematical Software (TOMS)* 29, 3 (2003), 245–262.

Rajaditya Mukherjee, Xiaofeng Wu, and Huamin Wang. 2016. Incremental deformation subspace reconstruction. In *Computer Graphics Forum*, Vol. 35. Wiley Online Library, 169–178.

Matthias Müller, Bruno Heidelberger, Matthias Teschner, and Markus Gross. 2005. Meshless deformations based on shape matching. In *ACM Trans. Graph. (TOG)*, Vol. 24. ACM, 471–478.

Przemyslaw Musialski, Christian Hafner, Florian Rist, Michael Birsak, Michael Wimmer, and Leif Kobbelt. 2016. Non-linear shape optimization using local subspace projections. *ACM Transactions on Graphics (TOG)* 35, 4 (2016), 1–13.

Vinod Nair and Geoffrey E Hinton. 2009. 3D object recognition with deep belief nets. *Advances in neural information processing systems* 22 (2009), 1339–1347.

Vinod Nair and Geoffrey E Hinton. 2010. Rectified linear units improve restricted boltzmann machines. In *ICML*.

HM Nasir. 2013. A new class of multicomplex algebra with applications. *Mathematical Sciences International Research Journal* 2, 2 (2013), 163–168.

CUDA Nvidia. 2008. Cublas library. *NVIDIA Corporation, Santa Clara, California* 15, 27 (2008), 31.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703* (2019).

Alex Pentland and John Williams. 1989. Good vibrations: Modal dynamics for graphics and animation. In *Proceedings of the 16th annual conference on Computer graphics and interactive techniques*. 215–222.

Eric Pesheck, Nicolas Boivin, Christophe Pierre, and Steven W Shaw. 2001. Nonlinear modal analysis of structural systems using multi-mode invariant manifolds. *Nonlinear Dynamics* 25, 1-3 (2001), 183–205.

Salah Rifai, Grégoire Mesnil, Pascal Vincent, Xavier Muller, Yoshua Bengio, Yann Dauphin, and Xavier Glorot. 2011a. Higher order contractive auto-encoder. In *Joint European conference on machine learning and knowledge discovery in databases*. Springer, 645–660.

Salah Rifai, Pascal Vincent, Xavier Muller, Xavier Glorot, and Yoshua Bengio. 2011b. Contractive auto-encoders: Explicit invariance during feature extraction. In *Icml*.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. 2015. Imagenet large scale visual recognition challenge. *International journal of computer vision* 115, 3 (2015), 211–252.

Sangriyadi Setio, Herlien D Setio, and Louis Jezequel. 1992. Modal analysis of nonlinear multi-degree-of-freedom structures. *IJAEM* 7, 2 (1992), 75–93.
Ahmed A Shabana. 2003. *Dynamics of multibody systems.* Cambridge university press.
Richard Socher, Jeffrey Pennington, Eric H Huang, Andrew Y Ng, and Christopher D Manning. 2011. Semi-supervised recursive autoencoders for predicting sentiment distributions. In *Proceedings of the 2011 conference on empirical methods in natural language processing*. 151–161.
Rasmus Tamstorf, Toby Jones, and Stephen F McCormick. 2015. Smoothed aggregation multigrid for cloth simulation. *ACM Trans. Graph. (TOG)* 34, 6 (2015), 245.
Adrien Treuille, Andrew Lewis, and Zoran Popović. 2006. Model reduction for real-time fluids. *ACM Transactions on Graphics (TOG)* 25, 3 (2006), 826–834.
Christoph Von-Tycowicz, Christian Schulz, Hans-Peter Seidel, and Klaus Hildebrandt. 2015. Real-time nonlinear shape interpolation. *ACM Transactions on Graphics (TOG)* 34, 3 (2015), 1–10.
Huamin Wang, James F O'Brien, and Ravi Ramamoorthi. 2011. Data-driven elastic models for cloth: modeling and measurement. *ACM transactions on graphics (TOG)* 30, 4 (2011), 1–12.
Huamin Wang and Yin Yang. 2016. Descent methods for elastic body simulation on the GPU. *ACM Trans. Graph. (TOG)* 35, 6 (2016), 212.
Yu Wang, Alec Jacobson, Jernej Barbič, and Ladislav Kavan. 2015. Linear subspace design for real-time shape deformation. *ACM Transactions on Graphics (TOG)* 34, 4 (2015), 1–11.
Steffen Wiewel, Moritz Becher, and Nils Thuerey. 2019. Latent space physics: Towards learning the temporal evolution of fluid flow. In *Computer Graphics Forum*, Vol. 38. Wiley Online Library, 71–82.
Stephen Wolfram et al. 1999. *The MATHEMATICA® book, version 4.* Cambridge university press.
Xiaofeng Wu, Rajaditya Mukherjee, and Huamin Wang. 2015. A unified approach for subspace simulation of deformable bodies in multiple domains. *ACM Transactions on Graphics (TOG)* 34, 6 (2015), 1–9.
Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. 2020. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems* (2020).
Hongyi Xu and Jernej Barbič. 2016. Pose-space subspace dynamics. *ACM Transactions on Graphics (TOG)* 35, 4 (2016), 1–14.
Hongyi Xu, Yijing Li, Yong Chen, and Jernej Barbič. 2015. Interactive material design using model reduction. *ACM Transactions on Graphics (TOG)* 34, 2 (2015), 1–14.
Yin Yang, Dingzeyu Li, Weiwei Xu, Yuan Tian, and Changxi Zheng. 2015. Expediting precomputation for reduced deformable simulation. *ACM Trans. Graph. (TOG)* 34, 6 (2015).
Yin Yang, Weiwei Xu, Xiaohu Guo, Kun Zhou, and Baining Guo. 2013. Boundary-aware multidomain subspace deformation. *IEEE transactions on visualization and computer graphics* 19, 10 (2013), 1633–1645.
Nianyin Zeng, Hong Zhang, Baoye Song, Weibo Liu, Yurong Li, and Abdullah M Dobaie. 2018. Facial expression recognition via learning deep sparse autoencoders. *Neurocomputing* 273 (2018), 643–649.
Jiayi Eris Zhang, Seungbae Bang, David I.W. Levin, and Alec Jacobson. 2020. Complementary Dynamics. *ACM Transactions on Graphics* (2020).
Yongning Zhu, Eftychios Sifakis, Joseph Teran, and Achi Brandt. 2010. An efficient multigrid method for the simulation of high-resolution elastic solids. *ACM Trans. Graph. (TOG)* 29, 2 (2010), 16.
Olgierd Cecil Zienkiewicz, Robert Leroy Taylor, Olgierd Cecil Zienkiewicz, and Robert Lee Taylor. 1977. *The finite element method.* Vol. 36. McGraw-hill London.

A FIRST-, SECOND-, AND THIRD-ORDER DERIVATIVE OF NONLINEAR ACTIVATION

In this appendix, we give the analytic formula for first- and second-order derivative of the activation function $\sin(\cdot)$ used in our network. We give partial formulation for the third-order derivative too, which is quite verbose. For the completeness, we move the entire formulation into the supplementary document.

The first derivative of $\sin(\cdot)$ under (first-order) CSFD is:

$$\sin(a + bi_1) = \sinh(a)\cos(b) + \cosh(a)\sinh(b)i_1.$$

The second-order CSFD perturbation of $\sin(\cdot)$ can be written as: $\sin(a + bi_1 + ci_2 + di_1i_2) = a' + b'i_1 + c'i_2 + d'i_1i_2$, where

$$\begin{aligned}
a' &= \sin(a)\cosh(b)\cosh(c)\cos(d) - \cos(a)\sinh(b)\sinh(c)\sin(d),\\
b' &= \sin(a)\cosh(b)\sinh(c)\sin(d) + \cos(a)\sinh(b)\cosh(c)\cos(d),\\
c' &= \cos(a)\cosh(b)\sinh(c)\cos(d) + \sin(a)\sinh(b)\cosh(c)\sin(d),\\
d' &= \cos(a)\cosh(b)\cosh(c)\sin(d) - \sin(a)\sinh(b)\sinh(c)\cos(d).
\end{aligned}$$

Similarly, we write the third-order multicomplex perturbation of $\sin(\cdot)$ as:

$$\begin{aligned}
&\sin(a + bi_1 + ci_2 + di_1i_2 + ei_3 + fi_1i_3 + gi_2i_3 + hi_1i_2i_3) =\\
&\qquad a' + b'i_1 + c'i_2 + d'i_1i_2 + e'i_3 + f'i_1i_3 + g'i_2i_3 + h'i_1i_2i_3.
\end{aligned}$$

The coefficient a' of the real part is:

$$\begin{aligned}
a' = &-\sin(a)\cosh(b)\cosh(c)\cos(d)\cosh(e)\cos(f)\cos(g)\cosh(h)\\
&+\sin(a)\cosh(b)\cosh(c)\cos(d)\sinh(e)\sin(f)\sin(g)\sinh(h)\\
&+\sin(a)\cosh(b)\sinh(c)\sin(d)\cosh(e)\cos(f)\sin(g)\sinh(h)\\
&+\sin(a)\cosh(b)\sinh(c)\sin(d)\sinh(e)\sin(f)\cos(g)\cosh(h)\\
&-\cos(a)\sinh(b)\cosh(c)\cos(d)\cosh(e)\cos(f)\sin(g)\sinh(h)\\
&-\cos(a)\sinh(b)\cosh(c)\cos(d)\sinh(e)\sin(f)\cos(g)\cosh(h)\\
&-\cos(a)\sinh(b)\sinh(c)\sin(d)\cosh(e)\cos(f)\cos(g)\cosh(h)\\
&+\cos(a)\sinh(b)\sinh(c)\sin(d)\sinh(e)\sin(f)\sin(g)\sinh(h)\\
&+\cos(a)\cosh(b)\sinh(c)\cos(d)\sinh(e)\cos(f)\sin(g)\cosh(h)\\
&-\cos(a)\cosh(b)\sinh(c)\cos(d)\cosh(e)\sin(f)\cos(g)\sinh(h)\\
&-\cos(a)\cosh(b)\cosh(c)\sin(d)\sinh(e)\cos(f)\cos(g)\sinh(h)\\
&-\cos(a)\cosh(b)\cosh(c)\sin(d)\cosh(e)\sin(f)\sin(g)\cosh(h)\\
&+\sin(a)\sinh(b)\sinh(c)\cos(d)\sinh(e)\cos(f)\cos(g)\sinh(h)\\
&+\sin(a)\sinh(b)\sinh(c)\cos(d)\cosh(e)\sin(f)\sin(g)\cosh(h)\\
&+\sin(a)\sinh(b)\cosh(c)\sin(d)\sinh(e)\cos(f)\sin(g)\cosh(h)\\
&-\sin(a)\sinh(b)\cosh(c)\sin(d)\cosh(e)\sin(f)\cos(g)\sinh(h).
\end{aligned} \tag{23}$$

The exact formulation of b', c', d', e', f', g', and h' can be found in the supplementary document.