Accelerated Complex Finite Difference for Expedient Deformable Simulation

ANONYMOUS AUTHOR(S)
SUBMISSION ID: 134

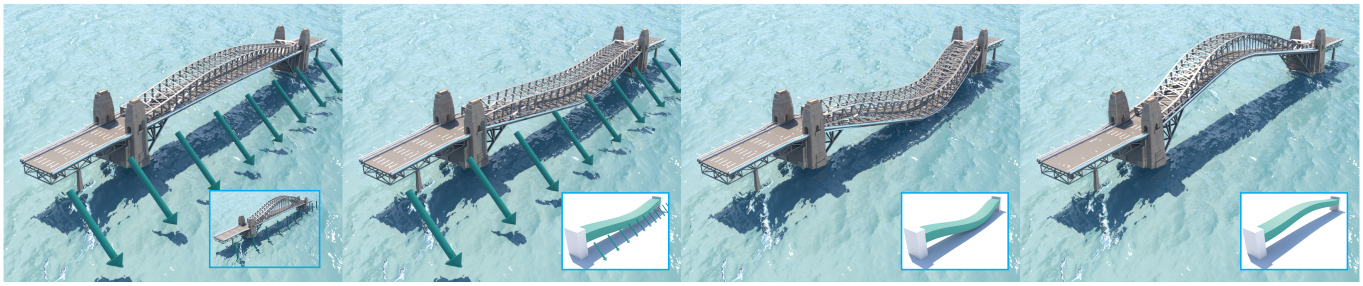

Fig. 1. We present an accelerated complex finite difference algorithm, which efficiently computes highly accurate numerical derivative. This method can be coupled with the Cauchy-Riemann formula to allow us to fully exploit existing (real-valued) linear algebra libraries to evaluate derivative of tensor-valued functions. This figure reports an example of designing the vibration frequency of a bridge model (21, 414 elements) by changing its geometry. The goal vibration frequency is visualized on a rectangular beam. Given an external force field, the bridge oscillates under the same frequency as the beam model does.

In deformable simulation, an important computing task is to calculate the gradient and derivative of the strain energy function in order to infer the corresponding internal force and tangent stiffness matrix. The standard numerical routine is the finite difference method, which evaluates the target function multiple times under a small real-valued perturbation. Unfortunately, the subtractive cancellation prevents us from setting this perturbation sufficiently small, and the regular finite difference is doomed for computing problems requiring a high-accuracy derivative evaluation. In this paper, we graft a new finite difference scheme, namely the complex finite difference (CFD), with physics-based animation. CFD is based on the complex Taylor series expansion, which avoids the subtraction for the first-order derivative approximation. As a result, one can use a very small perturbation to calculate the numerical derivative that is as accurate as its analytic counterpart. We significantly accelerate the original CFD method so that it is also as efficient as the analytic derivative. This is achieved by discarding high-order error terms, decoupling real and imaginary calculations, replacing costly functions based on the theory of equivalent infinitesimal, and isolating the propagation of the perturbation in composite/nesting functions. CFD can be further augmented with the multicomplex Taylor expansion and Cauchy-Riemann formula to handle higher-order derivatives and tensor-valued functions. We demonstrate the accuracy, convenience, and efficiency of this new numerical routine in the context of deformable simulation – one can easily deploy a robust simulator for general hyperelastic materials, including user-crafted ones to cater specific needs in different applications. Higher-order derivatives of the energy can be readily computed to construct modal derivative bases for reduced real-time simulation. Inverse simulation problems can also be conveniently solved using gradient/Hessian based optimization procedures.

0730-0301/2019/5-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

CCS Concepts: • **Computing methodologies** → **Physical simulation**; • **Mathematics of computing** → **Numerical analysis**; **Numerical differentiation**.

Additional Key Words and Phrases: Physics-based simulation, Deformable model, Numerical differentiation, Finite difference

ACM Reference Format:
Anonymous Author(s). 2019. Accelerated Complex Finite Difference for Expedient Deformable Simulation. *ACM Trans. Graph.* 1, 1 (May 2019), 16 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

As an essential computing task in physics-based simulation, the evaluation of various derivatives like total derivative, partial derivative, directional derivative, second- or high-order derivatives, etc. often stands out as a significant technical or implementation obstacle. Normally, people incline to infer an exact formula of derivative functions, which gives the best efficiency and accuracy. However, there are also many situations where a closed-form expression of the target function is not available (e.g. the function is given as a program sub-routine [Hahn et al. 2012]), or deriving its actual derivative is too involved for just performing preliminary proof-of-concept trials. The numerical derivative is then preferred.

The commonly used strategy for numerical derivative is the finite difference method. For instance, the *forward difference* scheme estimates the derivative as:

$$f'(x_0) = \lim_{\Delta x \to 0} \frac{f(x_0 + \Delta x) - f(x_0)}{\Delta x} \approx \frac{f(x_0 + h) - f(x_0)}{h}, \quad (1)$$

where a small perturbation $h \in \mathbb{R}$ is used to approximate $\lim_{\Delta x \to 0}(\cdot)$. It appears that the smaller h is, the better approximate Eq. (1) delivers. However, we are not allowed to make h arbitrarily small to improve the precision of Eq. (1). This is because the subtraction between two nearly equal numbers, such as $f(x_0 + h) - f(x_0)$ in

Eq. (1) when h is very small, could eliminate many of their significant digits and contaminate the result. This issue is often known as the *subtractive cancellation*. During the simulation, finite difference would accumulate numerical errors along the time integration and crash the solver quickly.

Fortunately, this numerical instability can be averted using the so-called *complex-step finite difference* [Abreu et al. 2018; Martins et al. 2003; Squire and Trapp 1998]. The trick is to apply the perturbation h in the imaginary domain after promoting f to be a complex function. The subtraction between the first-order terms is skipped in the complex Taylor series expansion [Lyness 1968], and we can make the perturbation h very small to accurately approximate the derivative without worrying about the subtractive cancellation. In this paper, we simply refer to this numerical differentiation method as the complex finite difference or CFD. CFD allows us to conveniently obtain a highly accurate numerical derivative without deriving its actual formulation, which could be otherwise tedious and error-prone.

On the downside, CFD has several fundamental limitations. First of all, promoting a real-valued function to be a complex-valued one induces significant computation overhead. A naïve CFD implementation as in existing literature [Martins et al. 2003; Squire and Trapp 1998] is often orders-of-magnitude slower than the analytic derivative. Secondly, complex-version Taylor expansion only circumvents the subtractive cancellation of the first-order derivative. Second- and higher-order derivatives still suffer with this issue and cannot be robustly approximated. In many simulation problems, we also need to deal with tensor functions, whose outputs are based on complicated numerical routines like Cholesky decomposition or SVD. Original CFD becomes awkward as those numerical procedures are difficult to be explicitly formulated. Promoting such tensor functions is rather involving, if not impossible. As an echo to those drawbacks, we augment the classic CFD scheme making it more efficient, generalizable, and robust. While our extensions utilize some known techniques like multicomplex number [Fleury et al. 1993; Price 1991] and Cauchy–Riemann equation [Ahlfors 1973], to the best of our knowledge, we are the first to introduce CFD to the graphics community, engineer it to be a handy off-the-shelf numerical solution to derivative evaluation for physics animation, and thoroughly validate its feasibility in the context of deformable simulation. Specifically, we summarize our contributions as follows:

- **Analysis** CFD is a relatively new numerical method, and it is not fully exploited by the graphics community yet. We provide an extensive explanation of its numerical mechanism, error source, and theoretical foundation.
- **Acceleration** We systematically optimize the original CFD scheme. Without losing accuracy, we obtain multifold speedups, and our accelerated CFD is as efficient as the analytic derivative. This is achieved by discarding high-order error terms, decoupling real and imaginary calculations, replacing expensive functions (e.g. trigonometric functions), and isolating the propagation of the perturbation in composite and nesting functions.
- **Adaptation** Instead of resorting to high-order Taylor expansion or Fourier expansion, we choose to promote a real-valued function with multicomplex arithmetic, which leads to a *multicomplex finite difference* scheme (MCFD) for high-order finite difference. Our acceleration techniques naturally synergize with this generalization without extra implementation efforts. In addition, we leverage the Cauchy-Riemann formula to further adapt CFD/MCFD for tensor-valued functions.
- **Application** We thoroughly validate CFD/MCFD in both numerical experiments as well as in complicated nonlinear deformable simulations. Without knowing the actual formulation of the internal force and the tangent stiffness matrix, nonlinear deformation can be robustly and accurately simulated with CFD/MCFD. First- and second-order modal derivative bases can also be constructed for sophisticated materials. Many challenging inverse simulation problems now can be easily tackled using the standard gradient/Hessian-based optimization approaches such as Newton's method (e.g. see Fig. 1).

2 RELATED WORK

Calculating the differentiation of a function is an important computational procedure in many graphics research problems. For instance, in physics-based animation [Witkin 1997], such as rigid body dynamics [Baraff 1989, 1991], fluid/smoke animation [Bridson 2015], and cloth simulation [Baraff and Witkin 1998; Goldenthal et al. 2007] etc, the key challenge is to solve the unknown ordinary/partial differential equations, and one needs to use numerical approaches to discretize the differential operation. In computational fabrication, the optimal design is often obtained via following the gradient direction of the inverse simulation [Chen et al. 2014; Schulz et al. 2017; Yan et al. 2018], not to mention a vast volume of research involving various optimization procedures, many of which rely on the information of the gradient and/or Hessian of the objective function.

Evaluating the derivative is also a key ingredient in deformable simulation [Terzopoulos et al. 1987] especially for hyperelastic models [Bonet and Wood 1997]. Those materials are fully characterized by the strain energy density $E(\mathbf{F})$ of the local deformation gradient $\mathbf{F}$. Modeling such materials requires the first- and/or second-order spatial derivatives of E to establish the equilibrium equation. Dynamic simulation can also be casted as an optimization problem of the *variational form* [Liu et al. 2013; Stern and Desbrun 2006]. Newton's method [Capell et al. 2002], quasi-Newton method [Liu et al. 2017] or gradient descent method [Wang and Yang 2016] can then be used when the gradient information is provided. The closed-form formulation of derivatives of E for some materials can be found in the literature [Bonet and Wood 1997; Sifakis and Barbic 2012; Smith et al. 2018]. However, for other more complicated models like phenomenological and user-crafted materials [Koyama et al. 2012; Martin et al. 2011], obtaining the analytic derivative is non-trivial and labor-intensive. For principal stretch based nonlinear materials, such as Ogden and spline-based materials [Xu et al. 2015], careful numerical thresholding is required, even at the rest configuration, to obtain the actual tangent stiffness matrix for implicit integration. Deriving those derivatives analytically could be tedious and seemingly unworthy, if the user just wants to toy with a new hyperelastic energy to see how it behaves in a given animation scenario. Even with the help of *symbolic differentiation* packages like `Mathematica` [Wolfram et al. 1996] and `Maple` [Maple 1994], the implementation efforts are still considerable. Besides, there are also many cases where the target function's formulation is not even accessible, and one has to use the numerical derivative to infer the underlying kinematics [Barbič et al. 2012; Hahn et al. 2012, 2013]. In model reduction, it is known that linear modes are not sufficient to capture large nonlinear deformation, and modal derivative [Barbič and James 2005; Yang et al. 2015] should be used. Those *derivative modes* are computed through evaluating the third-order gradient of the energy function (i.e. the Hessian of the internal force), which makes this technique less popular for more sophisticated materials other than the St. Venant-Kirchhoff (StVK) model.

The finite difference method is a standard procedure of computing the numerical derivative [Renardy and Rogers 2006]. Its variances include forward difference (FD), backward difference (BD), and central difference (CD). CD is twice expensive of FD or BD, but it is also the most accurate among them. Nevertheless, all of these schemes suffer from the subtractive cancellation issue: decreasing the magnitude of the perturbation does not make the finite difference converging, and the result will oscillate around the correct value and explode eventually [Brezillon et al. 1981]. This numerical behavior prevents the adoption of the finite difference method for applications that are sensitive to the accuracy of the differentiation.

On the other hand, CFD is a powerful finite difference scheme but often overlooked in classic numerical analysis textbooks [Squire and Trapp 1998]. It is sometimes also known as the complex-step derivative [Martins et al. 2003]. This method is based on the complex version of Taylor series expansion of the function, which dates back to the 1960s [Lyness 1967]. Unlike regular finite difference method, CFD obviates the subtractive cancellation problem (in the first-order approximation) so that a very small perturbation (e.g. 1×10^{-40} or even smaller) can be used making the resulting derivative approximation highly accurate. Indeed, we show that CFD is able to completely replace the analytic gradient without any accuracy concerns in deformable simulation. Due to its superior accuracy, CFD has been gradually recognized and used for the sensitivity analysis [Anderson et al. 2001; Butuk and Pemba 2003; Montoya et al. 2014; Voorhees et al. 2011]. For nonlinear finite element method (FEM) simulation with high-order shape functions, CFD has also been used to obtain numerical tangent stiffness matrix [Kim et al. 2011; Lebofsky 2013; Pérez-Foguet et al. 2000].

Because the target function is promoted to be a complex one, a naïve implementation of CFD involves much heavier computations than the real-valued finite difference. We show that this limitation can be ameliorated by carefully manipulating the promoted target function and discarding high-order perturbation terms. Our results show that we are able to achieve a multifold speedup, making CFD nearly as efficient as using the exact derivative. Instead of referring to the Fourier differentiation [Bagley 2006; Lai and Crassidis 2008], we use the multicomplex step finite difference [Lantoine et al. 2012] to handle high-order derivative. Doing so allows our acceleration scheme to be seamlessly integrated for high-order numerical derivatives.

CFD vs. automatic differentiation Another relevant and widely known differentiation technique is the automatic differentiation (AD) [Griewank and Walther 2008; Nocedal and Wright 2006; Rall 1981], which decomposes complicated functions with the *chain rule*. AD has been used in graphics [Grinspun et al. 2003; Guenter 2007; Mitchell and Hanrahan 1992]. Indeed, the back propagation optimization [Hecht-Nielsen 1992] commonly used from neural network training is a special implementation of the *reverse AD*.

A key difference between CFD and AD lies in the fact that *"AD uses exact formulas along with floating-point values"* [Neidinger 2010], and it is *"NOT numerical differentiation"* [Baydin and Pearlmutter 2014]. CFD, on the other hand, is a numerical approach seeking for the derivative approximate. AD is more sensitive to the smoothness of the function and could fail at discontinuities. CFD behaves more robustly in such cases: it always calculates the derivative as long as the target function exists. AD also has practical difficulties for high-order generalization [Margossian 2018]. For instance, some existing AD packages (e.g. `Adept` [Hogan 2014]) only deals with the first-order derivative. While one may perform first-order differentiation multiple times to obtain a high-order derivative. It has been argued that recursively applying AD leads to inefficient and numerically unstable code [Betancourt 2018; Margossian 2018]. High-order AD is seldom well supported and could be extremely slow. On the other hand, MCFD extension seamlessly generalizes our acceleration scheme to high-order cases with excellent robustness and accuracy. *Our accelerated CFD/MCFD is over 30× faster than commonly used AD packages even for the first-order case.* Tensor-valued functions that involving complicated numerical procedures are also problematic with AD. It remains unclear if the Cauchy–Riemann generalization [Ahlfors 1973] can be applied in AD. Our accelerated CFD is orthogonal to and complements the AD technique. Because CFD is highly accurate (as accurate as the analytic result), it is possible to deploy CFD/MCFD for calculating derivatives along the chain rule that could be otherwise troublesome to AD. More comprehensive comparisons between CFD/MCFD and AD can be found in Sec. 8.

3 BACKGROUND

In order to make the paper more self-contained, we start our discussion with a brief review of the error source of the finite difference method and the numerical issue of the subtractive cancellation.

Suppose that the function $f:\mathbb{R}\rightarrow\mathbb{R}$ is differentiable around $x=x_0$. After a small perturbation h is applied, it can be Taylor expanded as:

$$\begin{aligned} f(x_0+h) &= f(x_0)+f'(x_0)\cdot h+\frac{1}{2}f''(x_0)\cdot h^2+\cdots \\ &= f(x_0)+f'(x_0)\cdot h+\mathbf{O}(h^2), \end{aligned} \tag{2}$$

leading to the forward difference of Eq. (1). Eq. (2) also suggests that h should be as small as possible for a good approximation. In the meantime, because the total number of bits used to represent a real number is limited on a computer, all the floating-point arithmetics have the round-off error [Ueberhuber 2012], which is a small relative error also known as *machine epsilon* ϵ. For the double precision of `IEEE 754` floating-point standard [IEEE 1985], $\epsilon\approx1.11\times10^{-16}$. Normally, the round-off error does not seriously impair the stability or the accuracy of a numerical procedure. However, when h gets smaller, $f(x_0+h)$ and $f(x_0)$ become nearly equal to each other. Subtraction between them would eliminate many significant digits, and the result after the rounding could largely deviate from the actual value of $f(x_0 + h) - f(x_0)$.

We elaborate this issue using a simple decimal floating-point system with the precision of four. Here, a real number $a = 1999.99$ is represented as $\widetilde{a} = 1.999 \times 10^3$ because we only have four digits for the mantissa, and we use $\widetilde{(\cdot)}$ to denote a digitalized number in a floating-point system. In this example, we just choose the round-by-chop rule that discards all the out-of-precision digits, and the corresponding round-off error is:

$$E_{round} = \frac{|a - \widetilde{a}|}{|a|} = \frac{|1999.99 - 1.999 \times 10^3|}{|1999.99|} \approx 4.95 \times 10^{-4}. \quad (3)$$

Now, let $b = 1998.88$, which is represented as $\widetilde{b} = 1.998 \times 10^3$. The error of calculating $a - b$ with this toy floating-point system is:

$$\begin{aligned} E_{subtraction} &= \frac{|(\widetilde{a} - \widetilde{b}) - (a - b)|}{|a - b|} \\ &= \frac{|(1.999 - 1.998) \times 10^3 - (1999.99 - 1998.88)|}{|1999.99 - 1998.88|} \\ &\approx 0.1. \end{aligned} \quad (4)$$

We can see from Eqs. (3) and (4) that the rounding loses us the least important significant digit, and it only yields an error at the order of the floating-point precision (10^{-4}). However, the subtraction between $\widetilde{a}$ and $\widetilde{b}$ eliminates three leading significant digits, which yields a much more substantial error. If we set b even closer to a as $b = 1999.88$, $E_{subtraction}$ increases to 100% as all the significant digits are eliminated. This is why the cancellation of subtracting numbers of similar magnitude is also called *catastrophic cancellation*.

Some numerical literature (e.g. [Nocedal and Wright 2006]) shows that CD with the form of:

$$f'(x_0) \approx \frac{f(x_0 + h) - f(x_0 - h)}{2h}, \quad (5)$$

has a better accuracy with a quadratic error term of $\mathbf{O}(h^2)$, while FD and BD have an error term of $\mathbf{O}(h)$. This conclusion is based on the assumption that subtractive cancellation does not occur. As to be discussed in the next section, CD could be even more sensitive to smaller h (because of its faster convergent rate).

4 COMPLEX FINITE DIFFERENCE

CFD is based on the complex Taylor series expansion [Lyness 1968]. Let $(\cdot)^*$ denote a complex variable, and suppose $f^* : \mathbb{C} \rightarrow \mathbb{C}$ is differentiable around $x_0^* = x_0 + 0i$. If a perturbation h is applied at the imaginary domain, f^* can be expanded as:

$$\begin{aligned} f^*(x_0 + hi) &= f^*(x_0^*) + f^{*'}(x_0^*) \cdot hi + \frac{1}{2} f^{*''}(x_0^*) \cdot (hi)^2 \cdots \\ &= f^*(x_0^*) + f^{*'}(x_0^*) \cdot hi + \mathbf{O}(h^2). \end{aligned} \quad (6)$$

Since f^* is promoted from a real-valued function f, both $f^*(x_0^*) = f(x_0) \in \mathbb{R}$ and $f^{*'}(x_0^*) = f'(x_0) \in \mathbb{R}$ do not have imaginary parts. Taking the imaginary part of both sides of Eq. (6) leads to $\mathtt{Im}\big(f^*(x_0 + hi)\big) = \mathtt{Im}\big(f^*(x_0^*) + f^{*'}(x_0^*) \cdot hi\big) + \mathbf{O}(h^3)$, where $\mathtt{Im}(x + yi) = y \in \mathbb{R}$. We can then have the first-order CFD approximation as:

$$f'(x_0) = \frac{\mathtt{Im}\big(f^*(x_0 + hi)\big)}{h} + \mathbf{O}(h^2) \approx \frac{\mathtt{Im}\big(f^*(x_0 + hi)\big)}{h}. \quad (7)$$

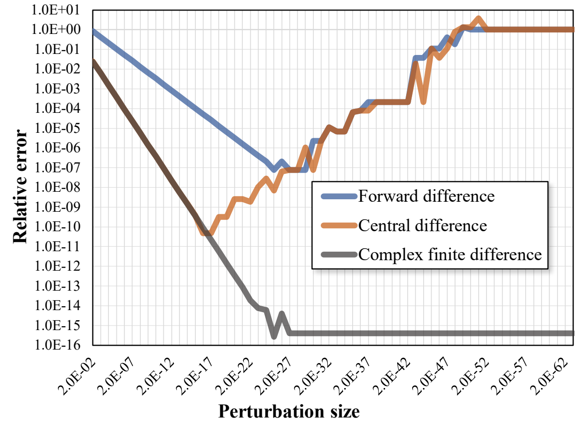

Fig. 2. We use FD, CD, and CFD to calculate the first-order derivative of $f(x) = e^x/(x^4 + x^2 + 1)$ at $x = 4$. The resulting numerical derivative is compared with the analytic derivative, and the relative error is plotted against the size of the perturbation, ranging from 2^{-2} to 2^{-63}.

Compared with Eq. (1) or Eq. (5), we can see that Eq. (7) does not have a subtractive numerator meaning it only has the round-off error regardless of the size of the perturbation h. In addition, the operation of $\mathtt{Im}(\cdot)$ removes the $(hi)^2$ term in Eq. (6), making the actual approximation error $\mathbf{O}(h^2)$. Thus, we can employ a very small h to obtain a highly accurate numerical derivative.

Fig. 2 reports a numerical experiment of $f(x) = e^x/(x^4 + x^2 + 1)$. We compute the first-order numerical derivative at $x = 4$ using FD, CD and CFD. The analytic derivative of this simple function can be easily derived as: $f'(x) = (x^4 - 4x^3 + x^2 - 2x + 1)e^x/(x^4 + x^2 + 1)^2$, and $f'(4) = 0.006593183194438$ is considered as the ground truth. In this example, both $f(x)$ and $f'(x)$ are well scaled, and the issue of subtractive cancellation starts to take place when $h \approx 2^{-26} \sim 1.0 \times 10^{-8}$ with FD and $h \approx 2^{-17} \sim 1.0 \times 10^{-6}$ with CD. We do see CD converges faster than FD before hitting the threshold of subtractive cancellation. However, both CD and FD explode quickly after the cancellation occurs. When h becomes smaller than $2^{-47} \sim 1.0 \times 10^{-15}$, the subtractive cancellation eliminates all the significant digits making $f(x_0 + h) - f(x_0)$ and $f(x_0 + h) - f(x_0 - h)$ vanished by the rounding. In this case, we cannot obtain any useful information of the derivative out of the finite difference approximation, and the relative error stays 100%. In real applications, f typically takes a high-dimension vector $\mathbf{x}$ as the dependable variable. f and $\partial f / \partial x_i$ may also be badly scaled. These circumstances make subtractive cancellation happen much earlier before h reaches $\sim 1.0 \times 10^{-6}$. As a result, the numerical derivative of FD and CD is highly fallible: a conservative h has a big approximation error, while an aggressive h could be even worse due to the cancellation. In physics-based simulations, FD/CD is always problematic and often the cause of the numerical instability. On the other hand, CFD shows a superior performance in terms of both convergence rate and numerical stability. As CFD does not have the subtractive cancellation problem, the relative error decreases consistently with smaller h. When h is sufficiently small (i.e. $h < 2^{-26} \sim 1.0 \times 10^{-8}$), CFD delivers a result with a relative error below 1.0×10^{-15}. Note that the "ground truth" itself also has a round-off error at the order of 10^{-16}. In other words, CFD is as accurate as the analytic derivative for sufficiently small h.

Complex function promotion In order to apply CFD, we must promote the real function $f(x)$ to be a complex one $f^*(x^*)$. While the specific form of $f(x)$ could be complicated, it is always constructed with binary operators of $+$, $-$, $\times$, $\div$ and unary operators including power function (x^a), exponential function (e^x), logarithmic function ($\ln x$), and trigonometric functions ($\sin x$ etc.). The promotion of these elementary functions follows the standard complex number arithmetic [Ablowitz and Fokas 2003]. For a quick reference, we also list the complex promotion of some commonly used functions in Appendix A.

If the efficiency is not the primary concern, CFD can be quickly implemented via overloading existing floating-point arithmetic operators with the corresponding complex version. `C++ Template` provides a flexible mechanism for this purpose: one can code $f(x)$ using a generic data type and choose the complex-type template specialization when CFD is needed, or the real-type template specialization if only the value of $f(x)$ suffices. Standard `C++ STD` library has a collection of stable complex number routines. Besides, there are also a few third-party opensource complex number libraries such as `Boost` [Schäling 2011] and `Eigen` [Guennebaud et al. 2014]. Nevertheless, such naïve CFD implementation induces a significant computation overhead. In many cases, CFD runs orders-of-magnitude slower than the analytic derivative. One of our major contributions is to optimize the CFD computation to regain the efficiency of the finite difference. This is to be detailed in the next section.

5 CFD ACCELERATION

Using general-purpose complex number arithmetic to promote $f(x)$ is actually "overkill" for just computing CFD for numerical derivatives. We show that CFD approximation can be substantially simplified and accelerated, and our accelerated CFD is as efficient as using the analytic derivative. Our strategy is based on the following three important observations:

- According to Eq. (7), it is clear that calculating the real part of f^* is unnecessary for CFD, therefore nearly half of computation brought by the complex promotion can be discarded.
- CFD complex number arithmetic is quite different from a general complex operation. The imaginary part of f^* comes from the applied perturbation hi, which is a very small value (i.e. $h < 1.0 \times 10^{-20}$). Many calculations can be simplified by treating h as an infinitesimal: for instance we can have $\sin h \sim h$ to avoid the expensive evaluation of the trigonometric function of $\sin h$.
- Because h appears as the denominator of Eq. (7), all the quadratic or higher-order terms of h in $\mathtt{Im}\big(f^*(x_0 + hi)\big)$ can be discarded, which only leads to an approximation error up to $\mathbf{O}(h)$.

5.1 Accelerate CFD of a Single Elementary Function

We start our discussion by assuming that $f(x)$ is an elementary function (i.e. listed in Appendix A). We take $f(x) = x^{1/m}$ an example to show how it can be much more efficiently evaluated for CFD.

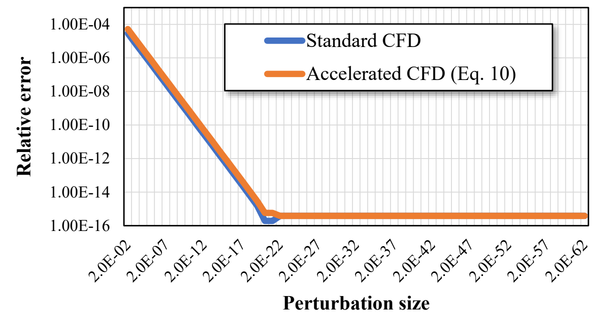

Fig. 3. Our fast CFD implementation has good numerical stability and accuracy. The relative error converges as quickly as the regular CFD and remains at the order of machine epsilon after h is sufficiently small.

First, the standard complex promotion (Eq. (44)) gives us:

$$\frac{\mathtt{Im}\big(f^*(x_0 + hi)\big)}{h} = \frac{1}{h}\left(r^{\frac{1}{m}} \cdot \sin\frac{\phi}{m}\right). \tag{8}$$

Here, $r(\cos\phi + \sin\phi i)$ is the polar form of $x_0 + hi$. Recalling that h is a very small quantity, we have:

$$\sin\phi = \frac{h}{r} \Rightarrow \phi = \frac{h}{r}, \tag{9}$$

because $\langle \sin a \sim a \rangle$ is a pair of equivalent infinitesimals when $a \to 0$. With Eq. (9), the RHS of Eq. (8) can be greatly simplified as:

$$\frac{1}{h}\left(r^{\frac{1}{m}} \cdot \sin\frac{\phi}{m}\right) = \frac{1}{h}\left(r^{\frac{1}{m}} \cdot \frac{\phi}{m}\right) = \frac{1}{h}\left(r^{\frac{1}{m}} \cdot \frac{h}{rm}\right) = \frac{r^{\frac{1}{m}}}{rm}. \tag{10}$$

Eq. (44)	Eq. (8)	Eq. (10)	Analytical
13.1 s	9.49 s (1.4×)	0.056 s (233×)	0.064 s

Table 1. Time statistics of using the optimized CFD formulations (i.e. Eqs. (8) and (10)) and the naïve CFD implementation (Eq. (44)) of the exponential function $f(x) = x^{1/m}$ for 100 million times. The computation time using analytical derivative is also reported for the reference. Our CFD simplification is over 200× faster than the naïve implementation. In this example, it is even faster than using the analytical derivative. The relative error is at the order of the machine epsilon (10^{-16}).

The performance improvement of Eq. (10) is substantial. We record the computation time of running Eqs. (44), (8), and (10) respectively as well as directly calculating the analytical derivative of $\left(x^{\frac{1}{m}}\right)' = \frac{1}{m}x^{\frac{1}{m}-1}$ for 100 million times on an `Intel i7` laptop. The result is reported in Tab. 1. As expected, we can see from the table that Eq. (8) modestly improves the calculation efficiency by discarding the real part computation. The most significant speedup originates from the equivalent infinitesimal based simplification, which frees us from performing the expensive trigonometric function calculation. In this example, CFD is even faster than using the analytic derivative because Eq. (10) has a simpler exponential term of $(\cdot)^{\frac{1}{m}}$ than the exponential term in the analytic derivative: $(\cdot)^{\frac{1}{m}-1}$. Meanwhile, our accelerated CFD retains all the favored advantages of CFD. As shown in Fig. 3, fast CFD implementation has the same convergency and accuracy. For small h, the relative error reaches the machine epsilon stably. Interestingly, if one further simplifies r such that $r = \sqrt{x_0^2 + h^2} = x_0$ when $h \to 0$, Eq. (10) converges to the analytic derivative formulation. This finding reveals that, unlike regular finite difference, *the actual derivative of the function is essentially hidden in its complex promotion.* This is another important reason that explains why CFD is able to achieve such high accuracy.

The strategy of the leveraging equivalent infinitesimals can be readily applied to all the elementary functions. For instance for trigonometric functions, the most expensive arithmetic is the evaluation of $e^{\pm h}$. Again, because h is an infinitesimal, we exploit the fact that $\langle e^{\pm h} \sim 1 \pm h \rangle$ is also a pair of equivalent infinitesimals. This simplification brings another orders-of-magnitude speedup.

5.2 Accelerate CFD of Composite Binary Operators

In reality, $f(x)$ houses a chain of binary operators such that:

$$f(x) = f_1(x) \circ f_2(x) \circ f_3(x) \circ \cdots \circ f_k(x) \circ \cdots \circ f_N(x), \tag{11}$$

for $\circ \in \{+, -, \times, \div\}$. Each $f_k(x)$ could also be a nesting composite of multiple unary functions: $f_k(x) = f_{k,1}(f_{k,2}(f_{k,3}(\ldots)))$. We defer the discussion of nesting operators to the next subsection and assume that the promoted form of each function along the chain is known.

Eq. (11) may be split into several sub-chains according to different parenthesizations and the operator priority. For instance, the example used in Fig. 2 can be understood as $f(x) = f_1(x)/\big(f_2(x) + f_3(x) + f_4(x)\big)$, where $f_1(x) = e^x$ is an exponential function; $f_2(x) = x^4$ and $f_3(x) = x^2$ are power functions; and $f_4(x) = 1$ is a constant. If a sub-chain only consists of addition and subtraction operators, which are independent for real and imaginary parts, we just evaluate the imaginary part of each promoted function f_k^* along the chain for CFD approximation and ignore the calculation for the real part.

However, if the sub-chain is concatenated with multiplication and/or division operators, we cannot simply discard the real part of each function because the real and imaginary parts are coupled in the multiplication operation – one can easily verify that the imaginary part of $f_1^*(x^*) \cdot f_2^*(x^*)$ contains the information of both real and imaginary parts of $f_1^*(x^*)$ and $f_2^*(x^*)$. Division is similar, which is regarded as the multiplication of the conjugate of the dividend.

We show that evaluating a multiplication chain can also be significantly accelerated based on the binary branching concisely encoded with a binary number. Let each promoted function on the chain $f_k^*(x^*) = a_k + b_k$, where the second addend b_k is an imaginary quantity. Our base case is the chain of a single promoted function $f^*(x^*) = f_1^*(x^*) = a_1 + b_1$ with two addends. Putting an additional multiplying function after it leads to $f^*(x^*) = f_1^*(x^*) \cdot f_2^*(x^*) = (a_1 + b_1)a_2 + (a_1 + b_1)b_2$. In other words, each item of a_1 and b_1 is multiplied by a_2 and b_2 respectively. The multiplication of $f_2^*(x^*)$ thus doubles the total number of addends. This procedure can also be visualized with a binary tree shown in Fig. 4. Each complex function $f_k^*(x^*)$ along the chain increments the height of the tree by one, and we have 2^N addends at the bottom level for a chain of N functions. Recall that imaginary parts of b_k correspond to a very small perturbation $b_k = hi \sim 0$, and we can discard all addends that are quadratic or higher-order of b_k. The key question here is how can we directly identify those addends without actually expanding the multiplication chain?

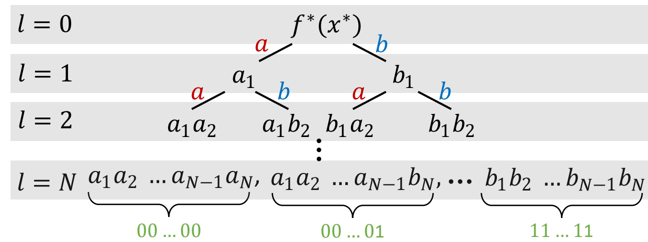

Fig. 4. The procedure of evaluating a chain of multiplications (and divisions) can be visualized with a binary tree. The leaf nodes can be concisely encoded by a binary number. As a result, we can discard higher-order infinitesimals that have two or more 1s (e.g. 011) at the bottom level.

From Fig. 4, we can see that each extra multiplication induces a binary branch towards the next level. A left branch appends an a_k after an existing addend while a right branch appends a b_k. The final form of a leaf addend depends on how many left and right branches at which levels it takes along the path from the root. Clearly, the leftmost and rightmost leaves are always $a_1 a_2 \ldots a_{N-1} a_N$ and $b_1 b_2 \ldots b_{N-1} b_N$. The second leftmost leaf differs from the leftmost one because it takes a right branch at the last level. Accordingly, its final form becomes $a_1 a_2 \ldots a_{N-1} b_N$. Interestingly, this branching mechanism mimics the ripple-carry addition of binary numbers. If we encode a_k with 0 and b_k with 1, all the addends at the bottom level, from left to right, can be concisely represented as a sequence of binary numbers $B_0, B_1, B_2, \ldots, B_{2^N-1}$ such that $B_k = (k)_{binary}$ is the binary representation of the decimal index k. For instance, if we have three functions along the chain, the eight leaf addends from B_0 to B_7 are: 000, 001, 010, 011, 100, 101, 110 and 111. The number of ones in B_k implies the order of h. Since anything higher-order than h^2 can be safely discarded, we only sum up addends with exact one 1-digit (the leftmost leaf is a real number, which is also discarded) such that: $\mathtt{Im}\big(f^*(x^*)\big) = a_1 a_2 b_3 + a_1 b_2 a_3 + b_1 a_2 a_3 + \mathbf{O}(h^2)$. From Eq. (6), we can also understand that $f(x_0) = \mathtt{Re}\big(f^*(x_0 + hi)\big) + \mathbf{O}(h^2)$ meaning replacing a_k by $f_k(x)$ only induces an approximation error of $\mathbf{O}(h^2)$. As a result, we can stick with our acceleration strategy of ignoring the real part of a promoted function. In practice, we also pre-compute the product among all the a_k as:

$$A = \Pi_{k=1}^{N} a_k = \Pi_{k=1}^{N} a_k f_k(x) + \mathbf{O}(h^2). \tag{12}$$

Therefore, a leaf node, say $a_1 b_2 a_3$ for instance, can be efficiently computed at $\mathbf{O}(1)$ time as:

$$a_1 b_2 a_3 = \mathtt{Re}\big(f_1^*(x^*)\big) \cdot \mathtt{Im}\big(f_2^*(x^*)\big) \cdot \mathtt{Re}\big(f_3^*(x^*)\big) \approx \frac{A}{f_2(x)} \cdot \mathtt{Im}\big(f_2^*(x^*)\big). \tag{13}$$

The timing benchmark shows that our strategy brings CFD approximation an additional 5× boost. After the value of each $f_k(x)$ is computed, the naïve implementation uses 21 *ms* to calculate the first-order CFD derivative for $N = 100$, while our method only needs 4 *ms* on an `i7` laptop.

5.3 Accelerate CFD of Composite Unary Operators

Real-world functions may also be in the nesting form of multiple unary operators:

$$f(x) = f_N(f_{N-1}(f_{N-2}(\cdots f_2(f_1(x))))), \tag{14}$$

where each f_k for $1 \le k \le N$ could be a power, exponential, logarithmic, or trigonometric function. We stick with the notation of $f_k^*(x^*) = a_k + b_k$, where b_k is an imaginary value, and $x_0 + hi = a_0 + b_0$ is the input of f_1^*, the innermost function. Note that the CFD approximation of $f'(x)$ is actually the ratio between b_N and b_0:

$$f'(x) \approx \frac{\mathtt{Im}\left(f_N^*\right)}{h} = \frac{\mathtt{Im}\left(f_N^*\right)i}{hi} = \frac{\mathtt{Im}\left(a_N + b_N\right)i}{hi} = \frac{b_N}{b_0}. \tag{15}$$

Similar to the multiplication case, the real and imaginary parts of an outer function are also coupled with the input real and imaginary parts from its inner function. The algebraic relation between b_N and b_0 could be complicated, and expanding the entire composite equation to compute the actual value of b_N is expensive.

Fortunately we notice that in order to compute $f'(x)$ with CFD, only the ratio between b_N and b_0 is needed, and their exact values are of less interest. Therefore, we convert b_N/b_0 to be:

$$\frac{b_N}{b_0} = \frac{b_1}{b_0} \cdot \frac{b_2}{b_1} \cdot \frac{b_3}{b_2} \cdot \ldots \cdot \frac{b_N}{b_{N-1}}. \tag{16}$$

A multiplicand in RHS of Eq. (16), b_k/b_{k-1}, describes how the imaginary perturbation is changed through f_k^*. An important observation here is *the imaginary part of a promoted function remains infinitesimally small after being applied with an infinitesimal imaginary perturbation.* This can be easily verified by the complex Taylor expansion: $f^*(x_0 + hi) \approx f^*(x_0^*) + f^{*'}(x_0^*) \cdot hi$, which leads to $\mathtt{Im}\left(f^*(x_0 + hi)\right) \approx f'(x_0) \cdot h = \mathbf{O}(h) \sim h$. In other words, all the b_k in Eq. (16) are small imaginary perturbations of the same order of hi. Therefore, we re-set each intermediate perturbation of b_{k-1} as h. In the meantime, its real part input a_{k-1} can be efficiently computed as f_{k-1} without resorting to f_{k-1}^* as:

$$\begin{aligned} \frac{b_k}{b_{k-1}} &= \frac{\mathtt{Im}\left(f_k^*(a_{k-1} + b_{k-1})\right)i}{b_{k-1}} \\ &\approx \frac{\mathtt{Im}\left(f_k^*(a_{k-1} + hi)\right)}{h} \approx \frac{\mathtt{Im}\left(f_k^*(f_{k-1} + hi)\right)}{h}. \end{aligned} \tag{17}$$

Eq. (17) literally breaks the coupling of the imaginary parts along the nesting chain – when computing b_k/b_{k-1}, the imaginary values from the inner functions are not required, and the propagation of the initial imaginary perturbation $b_0 = hi$ is isolated.

Discussion Eq. (16) should look immediately similar to the chain rule, which forms the foundation of AD techniques. Indeed, one may also understand Eq. (17) as breaking Eq. (14) using the chain rule and applying CFD to approximate each intermediate derivative afterwards (i.e. by setting $b_{k-1} = hi$ and $a_{k-1} = f_{k-1}$). In other words, Eq. (16) is practically equivalent to *augmenting AD with accelerated CFD* without referring to differentiation rules. Regular AD packages (e.g. `CppAD` [Bell 2012] and `Adept` [Hogan 2014]) mainly aim on first- or second-order derivatives, and their generalization to high-order derivative is nonintuitive and inefficient, if not impossible. However, as we will see in the next section, CFD can be elegantly generalized to handle high-order derivatives. All the acceleration techniques discussed in this section are naturally inherited.

6 MULTICOMPLEX PERTURBATION

Regular finite difference evaluates high-order derivative by recursively applying the first-order approximation of Eq. (1). For instance, the second-order derivative is approximated as:

$$f''(x_0) = \frac{f(x_0 + h) - 2f(x_0) + f(x_0 - h)}{h^2} + \mathbf{O}(h^2), \tag{18}$$

which requires two extra function evaluations for both $f(x_0 + h)$ and $f(x_0 - h)$. The complex Taylor series expansion of Eq. (6) gives a real second-order term (with a factor of i^2), which yields:

$$f''(x_0) = \frac{2\left(f(x_0) - \mathtt{Re}\left(f^*(x_0 + hi)\right)\right)}{h^2} + \mathbf{O}(h^2). \tag{19}$$

Eq. (19) only needs one extra function evaluation of $f^*(x_0 + hi)$: its imaginary part can be used for the first-order CFD while its real part is being used for the second-order CFD. However, both schemes suffer with the subtractive cancellation. Besides, computing $f^*(x_0 + hi)$ could be even slower than computing both $f(x_0+h)$ and $f(x_0-h)$ due to the extra complexity induced by the promotion. Therefore, second-order CFD is less appealing to us. A more numerical stable approach is based on *Fourier differentiation* [Bagley 2006], which generalizes the complex Taylor expansion to Fourier expansion by not retaining the perturbation on the imaginary axis:

$$f^*(x_0 + he^{\theta i}) = f^*(x_0^*) + f^{*'}(x_0^*) \cdot he^{\theta i} + f^{*''}(x_0^*) \cdot \frac{h^2}{2} e^{2\theta i} \cdots. \tag{20}$$

High-order derivative can be computed by using different argument angles of θ to cancel out unwanted terms. For instance, setting $\theta = \pi/4$ and $\pi + \pi/4$ leads to one possible second-order approximation [Lai and Crassidis 2008]:

$$f^{*''}(x_0^*) = \frac{\mathtt{Im}\left(f^*(x_0 + h \cdot i^{\frac{1}{2}}) + f^*(x_0 - h \cdot i^{\frac{1}{2}})\right)}{h^2} + \mathbf{O}(h^2). \tag{21}$$

While Fourier differentiation may be able to avoid the subtractive cancellation with carefully chosen θ, its formulation is quite different from the first-order CFD[1]. In practice, users have to use distinct implementations for different differentiation orders, and most calculations among them cannot be shared.

Alternatively, there is a more concise formula that generalizes the perturbation to be a *multicomplex* quantity, and we refer to this method as multicomplex finite difference (MCFD). The theoretic development of multicomplex algebra can be found in existing numerical analysis literature [Lantoine et al. 2012; Nasir 2013], which allows a complex number to have multiple imaginary directions. A multicomplex number is defined recursively: the base cases are the real number $\mathbb{R}$ and the regular complex number $\mathbb{C}$, which are considered as the zero and first order multicomplex sets $\mathbb{C}^0$ and $\mathbb{C}^1$. The complex number set $\mathbb{C}^1$ extends the real set by adding an imaginary unit i as: $\mathbb{C}^1 = \{x + yi | x, y \in \mathbb{C}^0\}$, and the multicomplex number up to an order of n is defined as:

$$\mathbb{C}^n = \left\{z_1 + z_2 i_n | z_1, z_2 \in \mathbb{C}^{n-1}\right\}. \tag{22}$$

[1] The fact is Fourier differentiation still has the subtractive cancellation issue. Explicitly avoiding the subtraction is not the real cure of the cancellation. This is out of scope of this paper, but numerical experiment clearly verifies this.

The order of a multicomplex number matches the number of its imaginary directions, and all the imaginary units i_n have the property of $i_n^2 = -1$. Fully expanding the recurrence of Eq. (22) yields:

$$
\begin{aligned}
\mathbb{C}^n = \quad & x_0 + x_1 i_1 + x_2 i_2 + \cdots + x_n i_n \\
& + x_{1,2} i_1 i_2 + \cdots + x_{n-1,n} i_{n-1} i_n \\
& + x_{1,2,3} i_1 i_2 i_3 + \cdots + x_{n-2,n-1,n} i_{n-2} i_{n-1} i_n \\
& \vdots \\
& + x_{1,2,\ldots,n} i_1 i_2 \cdots i_n,
\end{aligned}
\tag{23}
$$

where all of x_0, x_1, ..., x_n, $x_{1,2}$, $x_{2,3}$, ..., $x_{n-1,n}$, ..., $x_{1,2,\ldots,n}$ are real coefficients. For instance, setting $n = 2$ leads to $\mathbb{C}^2 = x_0 + x_1 i_1 + x_2 i_2 + x_{1,2} i_1 i_2$. A $\mathbb{C}^n$ number has 2^n x-coefficients: one x_0 for the real part, n coefficients x_1, x_2, ..., x_n for a single imaginary direction. All the other coefficients are for mixed imaginary directions with multiple i_j. Unlike quaternion [Shoemake 1985], the product between different imaginary units is commutative such that $i_j \cdot i_k = i_k \cdot i_j$ for $j \neq k$.

Following the derivation in [Lantoine et al. 2012], the Taylor series expansion of $f^\star$ under a multicomplex perturbation is:

$$
\begin{aligned}
& f^\star(x_0 + hi_1 + \cdots + hi_n) = f^\star(x_0) + f^{\star(1)}(x_0) \cdot h \sum_{j=1}^{n} i_j \\
& + \frac{f^{\star(2)}(x_0)}{2} \cdot h^2 \left(\sum_{j=1}^{n} i_j \right)^2 + \cdots + \frac{f^{\star(n)}}{n!} \cdot h^n \left(\sum_{j=1}^{n} i_j \right)^n + \cdots .
\end{aligned}
\tag{24}
$$

Here, $f^{\star(n)}$ is the n-th-order derivative of $f^\star$. $\left(\sum i_j\right)^k$ can be expanded following the *multinomial theorem*, and it contains products of mixed k imaginary directions for the k-th-order term. We refer the reader to [Lantoine et al. 2012; Nasir 2013] for a detailed step-by-step derivation. Because $\left(\sum i_j\right)^k \neq \left(\sum i_j\right)^l$ for $k \neq l$, Eq. (24) allows us to approximate an arbitrary-order derivative by directly extracting the corresponding imaginary combination, just as we did in CFD. In order to do so, $\mathtt{Im}(\cdot)$ should also be generalized to $\mathtt{Im}_\kappa(\cdot)$ to handle multiple imaginary directions:

$$
\mathtt{Im}_\kappa(z) = x_\kappa \in \mathbb{R}, \tag{25}
$$

which picks a coefficient x_κ that matches the imaginary combination of κ (i.e. the subscripts combination of i_j).

The MCFD approximation of the n-th-order derivative can then be concisely formulated as:

$$
f^{(n)}(x_0) = \frac{\mathtt{Im}^{(n)}\left(f^\star(x_0 + hi_1 + hi_2 + \ldots + hi_n)\right)}{h^n} + \mathbf{O}(h^2). \tag{26}
$$

Similarly, n-th-order partial derivative can be approximated as:

$$
\frac{\partial^n f(x_1, \cdots, x_p)}{\partial x_1^{k_1} \cdots \partial x_k^{k_p}} \approx \frac{\mathtt{Im}^{(n)}\left(f^\star(x_1 + h\sum_{j\in\Pi_1} i_j, \cdots, x_p + h\sum_{j\in\Pi_p} i_j)\right)}{h^n}, \tag{27}
$$

where $\mathtt{Im}^{(n)} = \mathtt{Im}_{1,2,\ldots,n}$ is a shortcut notation, which picks the coefficient of the mixed imaginary direction of $i_1 i_2 \cdots i_n$. $\Pi_j = \left\{ \sum_{l=1}^{j-1} k_l + 1, \cdots, \sum_{l=1}^{j} k_l \right\}$. By setting $n = 2$ in Eqs. (26) and (27), elements of the Hessian matrix (of a function $f(x, y) : \mathbb{R}^2 \rightarrow \mathbb{R}$) can be easily obtained as:

$$
\begin{cases}
\dfrac{\partial^2 f(x,y)}{\partial x^2} \approx \dfrac{\mathtt{Im}^{(2)}\left(f(x + hi_1 + hi_2, y)\right)}{h^2}, \\
\dfrac{\partial^2 f(x,y)}{\partial y^2} \approx \dfrac{\mathtt{Im}^{(2)}\left(f(x, y + hi_1 + hi_2)\right)}{h^2}, \\
\dfrac{\partial^2 f(x,y)}{\partial x \partial y} = \dfrac{\partial^2 f(x,y)}{\partial y \partial x} \approx \dfrac{\mathtt{Im}^{(2)}\left(f(x + hi_1, y + hi_2)\right)}{h^2}.
\end{cases}
\tag{28}
$$

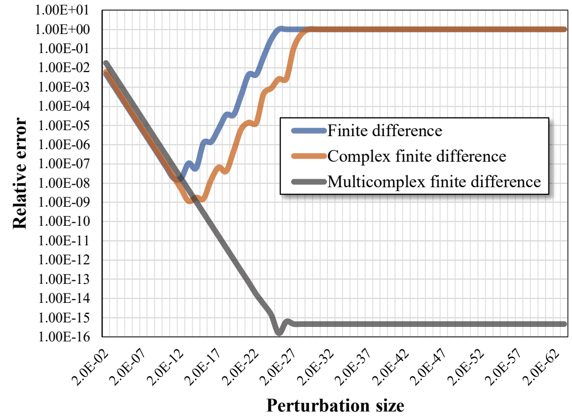

Fig. 5. The performance of MCFD approximation. We use the same test function of $f(x) = e^x/(x^4+x^2+1)$ as in Fig. 2 and calculate its second-order derivative at $x = 4$. The relative error w.r.t to value of analytic derivative is plotted against the size of the perturbation, ranging from 2^{-2} to 2^{-63}.

The most pleasing advantage of MCFD to us is its handy implementation. As long as CFD is implemented, all the routines for CFD can be recursively used for MCFD. More importantly, all the acceleration techniques discussed in Sec. 5 are also inherited with MCFD. The numerical performance of MCFD is excellent as reported in Fig. 5, where we evaluate the second-order derivative of $f(x) = e^x/(x^4 + x^2 + 1)$ at $x = 4$, the same example used in Fig. 2. The actual derivative $f''(x) = (x^8 - 8x^7 + 22x^6 - 12x^5 + 21x^4 - 12x^3 - 4x^2 - 4x - 1)e^x/(x^4 + x^2 + 1)^3$ is used as the reference. In this example, second-order finite difference (Eq. (18)) has a similar behavior as its first-order counterpart. After h hits a certain threshold ($\sim 1.0 \times 10^{-13}$), the subtractive cancellation makes the numerator a numerical zero leading to a 100% relative error. The second-order CFD approximation of Eq. (19) also suffers from this issue. MCFD however, accurately approximates the second-order derivative. With a sufficiently small h, the relative error becomes comparable to the machine epsilon, and the approximation can be used to fully replace the analytic derivative.

7 TENSOR FUNCTION

Most examples we have discussed so far is a real function taking a single real-valued input. In many simulation problems, however, we deal with functions with a tensor input. If we know how each component of the input tensor contributes to the output, we can simply overload the corresponding calculation to evaluate the promoted function value. For instance, $f(\mathbf{X}) : \mathbb{R}^{N\times N} \rightarrow \mathbb{R} = |\mathbf{X}|_F$ returns the Frobenius norm of the input matrix $\mathbf{X}$. As we know the exact form of this function is: $f(\mathbf{X}) = \sqrt{\sum\sum X_{i,j}^2}$, evaluating the partial derivative of $\partial f(\mathbf{X})/\partial X_{i,j}$ is really nothing more than fixing unrelated tensor components to pose f as a scalar-input function.

However, there are also many functions that do not rely on an explicit formulation such as the one that solves the input linear system:

$$f(\mathbf{X}) : \mathbb{R}^{N\times N} \rightarrow \mathbb{R}^N = \mathbf{X}^{-1}\mathbf{a}. \tag{29}$$

The exact inverse of a high-dimension matrix $\mathbf{X}$ is seldom given analytically. Instead, appropriate numerical routines like LU decomposition and forward-backward substitution are used to retrieve the function output. It is difficult for us to apply CFD or MCFD promotions without altering the underlying implementation of those numerical procedures.

An important advantage of CFD/MCFD is that one can exploit the *Cauchy-Riemann* (CR) formulation [Ahlfors 1973] to achieve (multi-)complex perturbations without overloading the complex arithmetic. CR form 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} following the definition of the multicomplex number (Eq. (22)) as:

$$z^n = z_0^{n-1} + z_1^{n-1} i_n \in \mathbb{C}^n = \begin{bmatrix} z_0^{n-1} & -z_1^{n-1} \\ z_1^{n-1} & z_0^{n-1} \end{bmatrix}. \tag{30}$$

Each of the 2×2 blocks in Eq. (30) 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.

CR form can also be generalized for tensors i.e. z_0^0 and z_1^0 can be real-valued tensor quantities. As a result, $f(\mathbf{X})$ of Eq. (29) can be promoted as:

$$f^*(\mathbf{X}^*) = \begin{bmatrix} \mathtt{Re}(\mathbf{X}^*) & -\mathtt{Im}(\mathbf{X}^*) \\ \mathtt{Im}(\mathbf{X}^*) & \mathtt{Re}(\mathbf{X}^*) \end{bmatrix}^{-1} \begin{bmatrix} \mathtt{Re}(\mathbf{a}^*) & -\mathtt{Im}(\mathbf{a}^*) \\ \mathtt{Im}(\mathbf{a}^*) & \mathtt{Re}(\mathbf{a}^*) \end{bmatrix}. \tag{31}$$

Because all the tensors are now real quantities, Eq. (31) can be evaluated without involving any complex number calculations. The resulting function value is also the CR form of $f^*(\mathbf{X}^*)$, and we can extract its imaginary values from off-diagonal blocks. Fig. 6 reports another numerical experiment of using CR form to calculate first-order and second-order derivative of the inverse of a 3×3 matrix: $f(\mathbf{X}) = \mathbf{X}^{-1} \in \mathbb{R}^{3\times 3}$ w.r.t $X_{2,2}$ (i.e. the element resides at the second row and second column of $\mathbf{X}$). In this example, we generate a random 3×3 non-singular matrix, and compute its inverse matrix analytically. The exact formulation of the first-order and second-order derivative of matrix inverse is:

$$\frac{\partial f}{\partial X_{2,2}} = -\mathbf{X}^{-1}\frac{\partial \mathbf{X}}{\partial X_{2,2}}\mathbf{X}^{-1}, \quad \frac{\partial^2 f}{\partial X_{2,2}^2} = -2\mathbf{X}^{-1}\frac{\partial \mathbf{X}}{\partial X_{2,2}}\mathbf{X}^{-1}\frac{\partial \mathbf{X}}{\partial X_{2,2}}\mathbf{X}^{-1},$$

and it is used as the reference.

The relative error of the numerical derivative computed using CR form as well as using the finite difference is plotted. We can see from Fig. 5 that CR form based CFD and MCFD also have an excellent accuracy, while the regular finite difference still suffers with the subtractive cancellation.

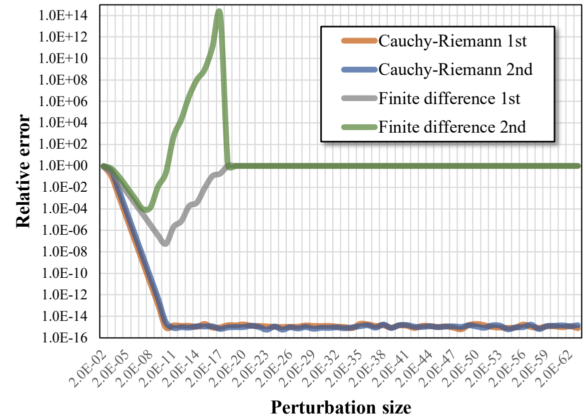

Fig. 6. Cauchy-Riemann formula allows us to use existing linear algebra libraries to compute high-order numerical derivative without referring to an explicit complex promotion. In this example, we compute the first- and second-order derivative of 3×3 matrix inverse. CR-form based CFD and MCFD still have excellent accuracy compared with regular finite difference.

8 EXPERIMENTAL RESULTS

We implemented CFD/MCFD on a desktop computer with an `Intel i7 8700K` CPU and 32 GB memory. Both regular complex arithmetic and the generalized multicomplex arithmetic were implemented using `C++ double` precision. The source code is also available in the supplementary material. While we believe CFD/MCFD will be useful for many graphics problems, in this paper we demonstrate its applications in modeling and simulating elastic objects. Our general observation is that one can fully rely on CFD/MCFD-based derivative without any accuracy concerns. Performance-wise, our accelerated CFD/MCFD is almost as efficient as analytic derivatives. We also compared CFD/MCFD with some widely used AD packages. While both CFD/MCFD and AD produce accurate results in well-conditioned problems, CFD/MCFD excels at its robustness for nonsmooth functions, high-order generalization, and tensor extension. Our accelerated CFD/MCFD is also much faster: it is over 20× faster than `C++` based AD packages and ∼ 300× faster than `Python` based AD packages.

	CFD	`Adept` (*s*)	`CppAD` (*s*)	`ADOL-C` (*s*)	`ad` (*s*)
1st	114 *ms*	11.1 (97×)	8.2 (72×)	1.4 (13×)	72.1 (632×)
2nd	242 *ms*	NA	11.2 (49×)	5.9 (24×)	80.3 (332×)
3rd	768 *ms*	NA	NA	51 (62×)	NA

Table 2. Time statistics of computing first- (**1st**), second- (**2nd**), and third-order (**3rd**) derivatives for 1 million times of the function: $f(x) = e^x/(x^4 + x^2 + 1)$ using CFD/MCFD and some popular AD packages.

Comparison with AD packages In the first experiment, we would like to examine the efficiency of our accelerated CFD/MCFD as well as some widely used AD packages including `Adept` [Hogan 2014], `CppAD` [Bell 2012], `ADOL-C` [Griewank et al. 1996], and `ad` [Lee

	CFD	Adept (s)	CppAD (s)	ADOL-C (s)	ad (s)
StVK 1st	9 ms	1.1 (122×)	0.8 (90×)	0.7 (78×)	7.1 (786×)
StVK 2nd	101 ms	NA	5.4 (54×)	5.2 (52×)	29 (288×)
NH 1st	12 ms	1.2 (99×)	0.8 (66×)	0.8 (65×)	7.2 (580×)
NH 2nd	117 ms	NA	5.6 (48×)	5.7 (49×)	31 (268×)

Table 3. Computing the internal force and tangent stiffness matrix for $10k$ linear tetrahedral elements of StVK and Neo-Hookean materials. Similar to Tab. 2, our accelerated CFD/MCFD is much faster than AD packages.

2013]. The first three libraries are in C++, and ad is a famous Python package. We record the time performance for evaluating derivatives (for 1 million times) of the function: $f(x) = e^x/(x^4 + x^2 + 1)$ at $x = 4$ (i.e. the one used in Figs. 2 and 5). The computation time for the analytic first- and second-order derivatives is 104 ms and 238 ms respectively, which is quite close to our CFD/MCFD taking 114 ms and 242 ms. This function is smooth and differentiable, and all AD packages return accurate first-order derivative results successfully. Yet, our method is massively faster than AD packages as reported in Tab. 2. In general, the accelerated CFD/MCFD is dozens times faster than C++ based AD packages and hundreds times faster than Python based ones. In this experiment, Adept does not support second-order derivative natively. CppAD only supports high-order derivative up to the second order. ADOL-C is the most sophisticated package, which has dedicated sub-routines for second-order and high-order derivatives. Nevertheless, it is still more than one order slower than our method. ADOL-C becomes even more slower for higher-order derivatives as it calls the first-order routine repeatedly for high-order cases (i.e. with its `forward()` routune). Python package is the slowest among all.

We also assess the robustness of AD packages for nonsmooth functions. Let $f(x) = \log^2\left(1 - \sqrt{(x-1)^2}\right)$. Its analytic first- and second-order derivative can be easily derived as:

$$f'(x) = -\frac{2(x-1)\log\left(1-\sqrt{(x-1)^2}\right)}{\left(1-\sqrt{(x-1)^2}\right)\sqrt{(x-1)^2}}, \tag{32}$$

and

$$f''(x) = -\frac{2\left(\log\left(1-\sqrt{(x-1)^2}\right)-1\right)}{\left(1-\sqrt{(x-1)^2}\right)^2}. \tag{33}$$

We notice that $x = 1$ is actually a singular point of the function. Without explicitly cancelling out $\sqrt{(x-1)^2}$ from Eqs. (32) and (33), AD packages that overload elementary arithmetic with differentiation rules tend to yield the division-by-zero error[2]. In this experiment, only Adept successfully returns the first-order derivative of this function, but It yields a #IND error for the second-order case. All other AD packages return either NaN, #IND or ZeroDivisionError error. On the other hand, CFD/MCFD robustly handle this function derivative without any special treatments.

[2]We may be able to avoid this numerical instability of AD by expanding and simplifying the derivative function. But if we choose to do so, we are literally deriving the analytic formula of the derivative function, and why do we bother to use AD?

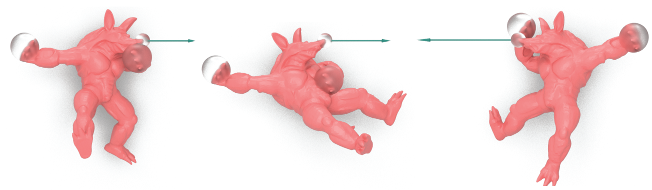

Fig. 7. We simulate a Neo-Hookean Armadillo model using Newton's method. The Armadillo has 69, 074 elements. The gradient and Hessian of the target function f (i.e. Eq. (35)) is approximated using numerical CFD/MCFD. The result is identical to the one computed using analytic gradient and Hessian.

We observe similar results when applying CFD/MCFD and AD in deformable simulation computations. Tab. 3 lists the time performance of computing the internal force and tangent stiffness matrix for $10k$ linear tetrahedral elements of StVK and Neo-Hookean materials, which are the first- and second-order partial derivatives of the energy function. The analytic formulations of those two energies are known. We use Vega library [Barbič et al. 2012] to compute the analytic force and stiffness matrix. For the StVK material, it takes 11.8 ms and 103.5 ms for the first- and second-order derivatives. For the Neo-Hookean material, the computation time is 12.1 ms and 112.5 ms respectively. This performance measure is very similar to our accelerated CFD and MCFD as shown in the table. In this experiment, most AD packages deliver correct results (expect for Adept) but they are all much slower than accelerated CFD and MCFD.

8.1 Application I: Accurate Nonlinear Optimization

Dynamic simulation of a deformable object requires solving a nonlinear system of the force equilibrium. For instance, the implicit Euler time integration scheme leads to:

$$\mathbf{M}(\mathbf{u}_{n+1} - \mathbf{u}_n - \Delta t\dot{\mathbf{u}}_n) = \Delta t^2(\mathbf{f}_{int}(\mathbf{u}_{n+1}) + \mathbf{f}_{ext}), \tag{34}$$

where $\mathbf{M}$ is the mass matrix. $\mathbf{f}_{int}$ and $\mathbf{f}_{ext}$ stand for the elastic internal force and the external force. The subscript $(\cdot)_n$ denotes the time integration step, and Δt is the time step size. $\mathbf{u}_{n+1}$ is the unknown displacement vector we want to compute. This equilibrium is often treated as an optimization problem known as its variational form [Liu et al. 2013; Stern and Desbrun 2006] of:

$$\arg\min_{\mathbf{u}} f(\mathbf{u}), \quad f(\mathbf{u}) = \frac{1}{\Delta t^2}\left\|\mathbf{M}^{\frac{1}{2}}(\mathbf{u} - \mathbf{u}^*)\right\|^2 + E(\mathbf{u}), \tag{35}$$

where $\mathbf{u}^* = \mathbf{u}_n + \Delta t\dot{\mathbf{u}}_n + h^2\mathbf{M}^{-1}\mathbf{f}_{ext}$ is a known vector. E is the nonlinear elastic energy. Eq. (35) can be solved using the classic Newton's method, which approximates $f(\mathbf{u})$ with a quadratic form and calculates an incremental improvement of $\Delta\mathbf{u}$ as $\Delta\mathbf{u} = -\mathbf{H}\cdot\partial f/\partial\mathbf{u}$. Matrix $\mathbf{H}$ is the Hessian matrix, and it is the second-order partial derivative of f: $\mathbf{H} = \partial^2 f/\partial\mathbf{u}^2$. We simulate nonlinear dynamics of a Neo-Hookean Armadillo (with 69, 074 elements) using Newton's method and drag its mouth back and forth. The gradient and Hessian of f are approximated with CFD/MCFD. The elastic energy density E of the Neo-Hookean material is

$$E_{NH} = \lambda(J-1)^2 + \mu(J^{-2/3}I_1 - 3), \tag{36}$$

Fig. 8. The Armadillo model falls quickly and hits a glassy rod. Due to the sharp collision, gradient descent method [Wang and Yang 2016] yields artifact because the residual is not sufficiently reduced. Regular finite difference method crashes instantly. Newton's method with MCFD-based Hessian yields the same result as using the analytic Newton. Newton-PCG with MFCD-based directional derivative also has the same result.

where $J = |\mathbf{F}|$ is the determinant of the deformation gradient $\mathbf{F}$, and $I_1 = \mathtt{tr}(\mathbf{F}^\top \mathbf{F})$. λ and μ are Lamé constants. In our CFD/MCFD implementation, we treat E as a nested composite function $E_{NH} = E_1(J(\mathbf{F})) + E_2(I_1(\mathbf{F}))$. Snapshots of the deformed Armadillo are reported in Fig. 7. This animation is *identical* to the one obtained using analytic gradient and Hessian.

Alternatively, one may also use the Newton-PCG method, which replaces the direct solver used at each Newton iteration with an iterative PCG solver. As explained in [Yang et al. 2015], each Newton-PCG iteration calculates the product of $\mathbf{K}|_{\mathbf{u}_0} \cdot \mathbf{p}$, where $\mathbf{K}|_{\mathbf{u}_0}$ is the current tangent stiffness matrix at $\mathbf{u} = \mathbf{u}_0$, and $\mathbf{p}$ is a known displacement vector. This product can also be understood as the directional derivative of the energy function E and be numerically computed via CFD as:

$$\mathbf{K}|_{\mathbf{u}_0} \cdot \mathbf{p} = \left.\frac{\partial^2 E}{\partial \mathbf{u}^2}\right|_{\mathbf{u}_0} \cdot \mathbf{p} = \nabla_{\mathbf{p}} E|_{\mathbf{u}_0} \approx \frac{\mathtt{Im}\big(E^*(\mathbf{u}_0 + h \cdot \mathbf{p}i)\big)}{h}. \tag{37}$$

As shown in Fig. 8, CFD-based directional derivative is also highly accurate, which produces the same result of analytic Newton and MCFD Newton. The regular finite difference crashes immediately when the Armadillo collides with the glassy rod.

8.2 Application II: Intuitive Hyperelastic Simulation

For hyperelastic models, the form of the elastic energy (i.e. Eq. (35)) solely determines the deformed shape given inertial and external forces. Hyperelastic energy is typically defined based on three *isotropic invariants* of the deformation gradient: $I_1 = \mathtt{tr}(\mathbf{F}^\top\mathbf{F})$, $I_2 = \mathtt{tr}\big((\mathbf{F}^\top\mathbf{F})^2\big)$, and $I_3 = |\mathbf{F}^\top\mathbf{F}|^2$. Intuitively, I_1 measures the length change of the deformation; I_2 measures the area change of the deformation; and I_3 measures the volume change of the deformation. As long as the internal force $\partial E/\partial \mathbf{u}$ and the tangent stiffness matrix $\partial^2 E/\partial \mathbf{u}^2$ are available, the dynamic behavior of the deformable body can be simulated using standard FEM. The closed-form formulation of $\partial E/\partial \mathbf{u}$ and $\partial^2 E/\partial \mathbf{u}^2$ for some material models such as co-rotational model, StVK model, Neo-Hookean model are available in the literature [Bonet and Wood 1997; Sifakis and Barbic 2012; Smith et al. 2018]. However, there are many other materials such as Fung, Mooney-Rivlin, Ogden, Yeoh, Arruda Boyce models or the more general Polynomial model. Their energy structure can be easily followed, but deriving the actual formulation of force and stiffness matrix prevents these materials from being more widely employed by the graphics community. CFD and MCFD allow us to conveniently simulate hyperelastic materials with light-weight implementation efforts. As reported in Fig 9, we simulate all of those materials using CFD/MCFD under standard bending, compressing, stretching and twisting tests. In this experiment, we use invertible StVK energy [Irving et al. 2004] to improve the stability of the regular StVK material. Timing information of different CFD/MCFD implementations is compared in Fig. 10.

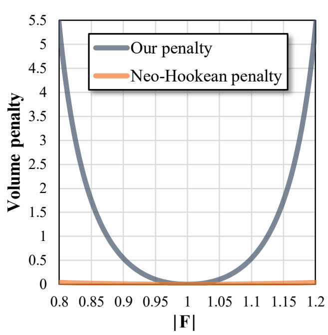

Fig. 11. We design a new volume penalty term of $\log^2\big(1 - 4(J-1)^2\big)$, which yields much bigger internal forces when $J = |\mathbf{F}|$ deviates from 1 than the regular Neo-Hookean volume penalty of $(J-1)^2$ does.

In many situations, the user wants to use customized materials for specific needs in an animation scenario. For instance Smith and colleague [2018] proposed a new Neo-Hookean-like hyperelastic energy for a stable integration and volume preservation under large deformation. Using CFD/MCFD, users can freely explore various such energy densities without tedious derivations for internal force and Hessian. For instance, we design a new hyperelastic model:

$$E_{volume} = \mu(J^{-2/3} I_1 - 3) + \frac{\lambda}{2}\log^2\left(1 - 4(J-1)^2\right). \tag{38}$$

As plotted in Fig. 11, E_{volume} triggers a much stronger resistance force to when $J = |\mathbf{F}|$ deviates from 1 and better preserves the volume (i.e. see Fig. 12). In this example, the rest-shape volume of the jelly box is 0.64. After compressing its height by 65%, the new volume of the jelly box becomes 0.63 with E_{volume} and 0.61 with the stable Neo-Hookean material [Smith et al. 2018]. While numbers look close, we can clearly see that the compressed box is wider spread with E_{volume}.

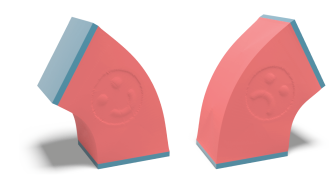

Fig. 13. Example-based hyperelastic energy can also be easily handled with CFD/MCFD. We make the energy be the function of bending angle so that a smiling face appears when the box bends to left and a sad face appears when the box bends to right. This box model has 14, 678 elements.

CFD/MCFD can deal with even more complicated energies. Another example is reported in Fig. 13. In this example, we use an example-based hyperelastic energy as in [Martin et al. 2011], which

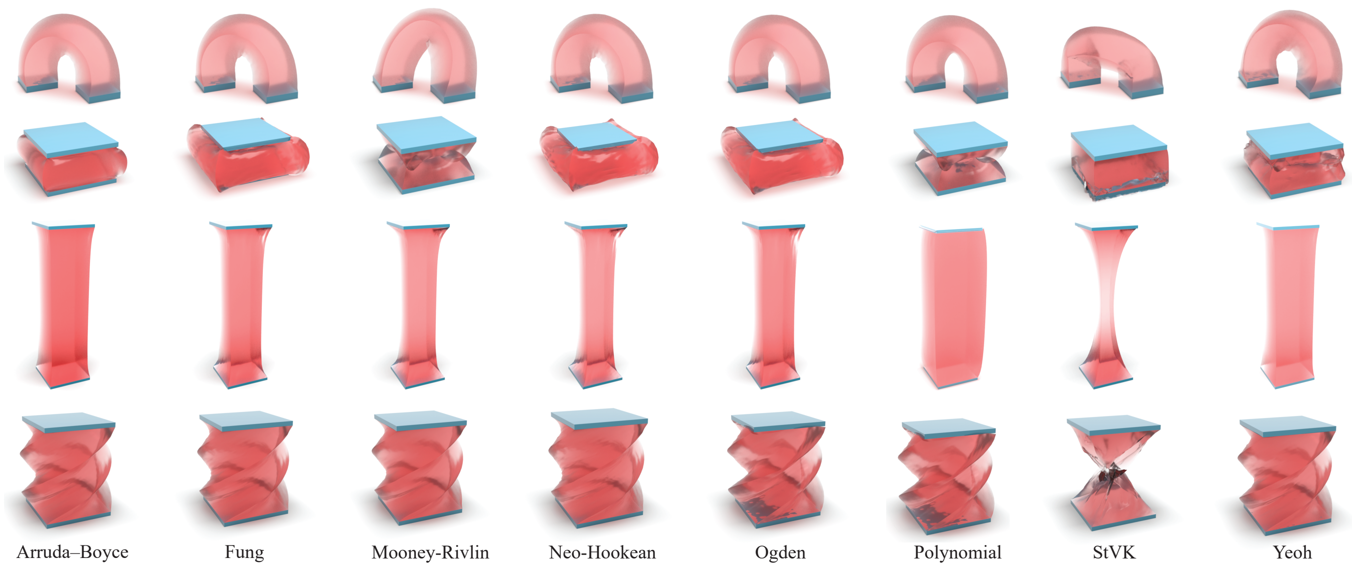

Fig. 9. CFD/MCFD allows the user to easily simulate all kinds of hyperelastic materials. The figure reports the material behaviors under standard bending, compressing, stretching, and twisting tests of a box model with 14, 678 elements. From left to right, each column gives the result of Arruda–Boyce, Fung, Mooney-Rivlin, Neo-Hookean, Ogden, Polynomial, invertible StVK [Irving et al. 2004] (for improved stability), and Yeoh materials.

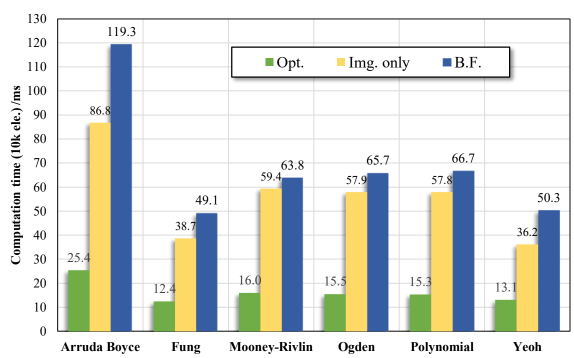

Fig. 10. Timing information of CFD/MCFD derivative in simulating various hyperelastic materials. **Opt.** is the optimized CFD/MCFD computation time. **Img. only** is the time without computing the real part of the promoted energy functions. **B.F.** is the computation time using a brute-force CFD/MCFD implementation.

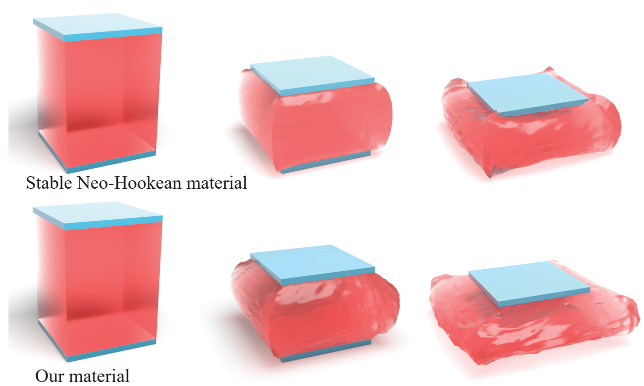

Fig. 12. Our new material (Eq. (38)) with a more aggressive volume penalty term is able to better preserve the volume of this jelly box during the compression than the stable Neo-Hookean material [Smith et al. 2018].

has two target shapes, each of which embeds a smiling face ☺ or a sad face ☹ on the surface. We design this energy to be the function of the bending orientation so that corresponding internal forces arise when the box is bent to a certain direction. CFD/MCFD frees us from formulating the animation system and to quickly toy with many of such examples to achieve more interesting animations.

For a customized material, it is possible that the user-crafted energy has some singularities due to its complex formulation. In this case, AD packages, regardless of their slow performance, could even fail the simulation if any element reaches the singularity. To better elaborate this, we create another energy with the form of:

$$E_{singular} = \mu(J^{-2/3}I_1 - 3) + \lambda(J-1)^2 + \sqrt{\cos^2 4(J-1)} - 1. \quad (39)$$

As shown in Fig. 14, if we slowly bend the dragon with the material of $E_{singular}$ using AD, the system crashes with the division-by-zero error when an element hits the singular point. CFD/MCFD is robust in such situations. Referring to Eq. (7), it is easy to see that as long as $f(x_0)$ exists, $f^*(x_0 + hi)$ also exists because it is promoted orthogonally towards the real domain. Therefore, CFD always returns a well-estimated derivative value because h is also nonzero.

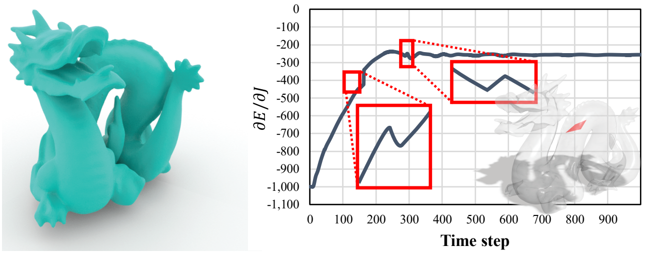

Fig. 14. Complicated energy formulation as Eq. (39) often hides singularities that are unfriendly for AD. CFD/MCFD can tackle this issue robustly.

8.3 Application III: Expressive Model Reduction

Model reduction is a widely-used technique to produce real-time deformable animation. This technique needs a pre-built subspace, which defines all the possible deformations of the deformable body. The standard method for subspace construction is based on the modal analysis [Pentland and Williams 1989], which provides the optimal vibrational modes around the rest shape. For nonlinear models, we need to compute *derivative modes* that first-order approximate a low-frequency nonlinear vibration [Barbič and James 2005; Yang et al. 2015]. Computing derivative modes requires the calculation of the force Hessian (i.e. the third-order derivative of E). Therefore, this powerful technique is normally used only for the StVK material, whose stiffness matrix is quadratic w.r.t to the displacement vector. Applying nonlinear model reduction to other materials using modal derivative is less exploited due to the barrier of computing high-order energy gradients. CFD/MCFD allows us to build expressive and compact subspace easily for any given hyperelastic material. Fig. 15 shows snapshots of a real-time simulation of six falling dinosaur models using 30 first-order modal derivatives. Each dinosaur model has 356, 48 elements, and they are of Fung, Mooney-Rivlin, Ogden, Yeoh, Arruda Boyce, and Polynomial materials. Yang and colleagues [2015] introduced a method that generalizes modal derivative to higher-order nonlinear shape approximation. This method can also be readily implemented with MCFD. As shown in Fig. 16, we apply a circular force to bow the dinosaur model. Second-order modal derivatives are able to capture extreme bending effects. In this experiment, the hyperelastic material of Eq. (38) has a strong volume preserving term, which prevents this material from being extremely bent as other materials under the same external forces.

8.4 Application VI: Convenient Inverse Design

A lot of design problems tweak a collection of parameters to make sure that the simulated result matches certain specific measures like the maximum stress, deflection magnitude and so on. While there are many techniques (e.g. the well-known adjoint method) that are capable of handling those problems, we show that CFD/MCFD is also a convenient alternative to deal with inverse simulations.

In Fig. 1, we show an example where the user wants to adjust the linear vibration frequencies of a bridge for a given external wind field by changing three primary geometry parameters: length l, width w and the height of the arch top t. For an intuitive visualization of a frequency pattern, our system allows the user to apply this wind field to a standard rectangular beam (with two ends fixed) and to change its geometry/material to generate a preferred vibration pattern (see Fig. 17). The principle vibration of a linear structure under a given direction $\mathbf{u}$ is described by the *Rayleigh quotient* defined as $\omega^2 = \mathbf{u}^\top\mathbf{K}\mathbf{u}/\mathbf{u}^\top\mathbf{M}\mathbf{u}$. The wind is modeled as an acceleration field $\mathbf{a}$ meaning $\mathbf{u} = \mathbf{K}^{-1}\mathbf{M}\mathbf{a}$. As a result, the frequency design procedure can be formulated as an optimization problem of:

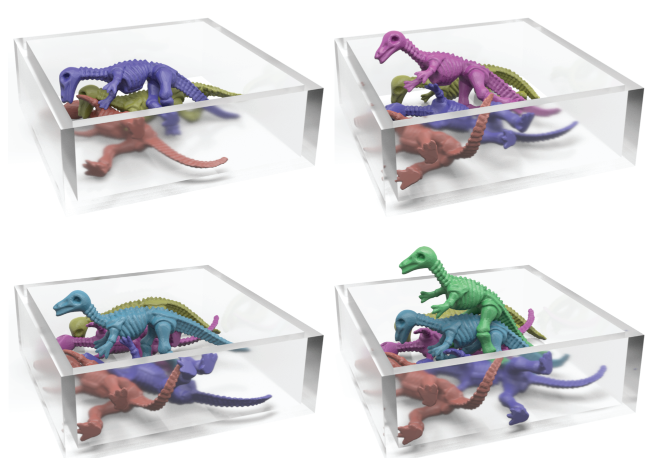

Fig. 15. Real-time simulation of six falling dinosaur models using modal derivative (30 modes for each dinosaur). The first-order derivative modes are computed using CFD, and we use Fung, Mooney-Rivlin, Ogden, Yeoh, Arruda–Boyce and Polynomial materials for each dinosaur.

$$\arg\min_{l,w,t} f, \quad f(l,w,t) = \left\| \omega^{*2} - \frac{\mathbf{a}^\top\mathbf{M}\mathbf{K}^{-1}\mathbf{M}\mathbf{a}}{\mathbf{a}^\top\mathbf{M}\mathbf{K}^{-1}\mathbf{M}\mathbf{K}^{-1}\mathbf{M}\mathbf{a}} \right\|^2, \quad (40)$$

where ω^{*2} is our target frequency. $\mathbf{M} = \mathbf{M}(l,w,t)$ and $\mathbf{K} = \mathbf{K}(l,w,t)$ are tensor functions of the unknown geometry parameters l, w, t to be optimized. In this example, we use the CR form (Eq. (31)) to promote $\mathbf{M}(l,w,t)$ and $\mathbf{K}(l,w,t)$, and Newton's method is used to solve Eq. (40). Thanks to the accurate Hessian obtained by MCFD, our solver quickly finds the optimal geometry only with few iterations.

9 CONCLUSION AND FUTURE WORK

In this paper, we show how to accelerate and generalize complex-step finite difference to accurately obtain a numerical derivative of an arbitrary order. Its superior precision comes from complex or multicomplex promotion of the target function, which avoids the subtractive cancellation issue in standard finite difference methods. Without worrying about losing many significant digits during the calculation, CFD and MCFD allow us to have a very small-size perturbation to obtain a numerical derivative at the precision of machine epsilon implying it is as accurate as the analytic derivative. We propose a collection of acceleration techniques that avoid redundant and costly calculations induced by the complex promotion. Therefore our CFD and MCFD are as efficient as analytic derivative, but we are freed from the derivation for the actual differentiation.

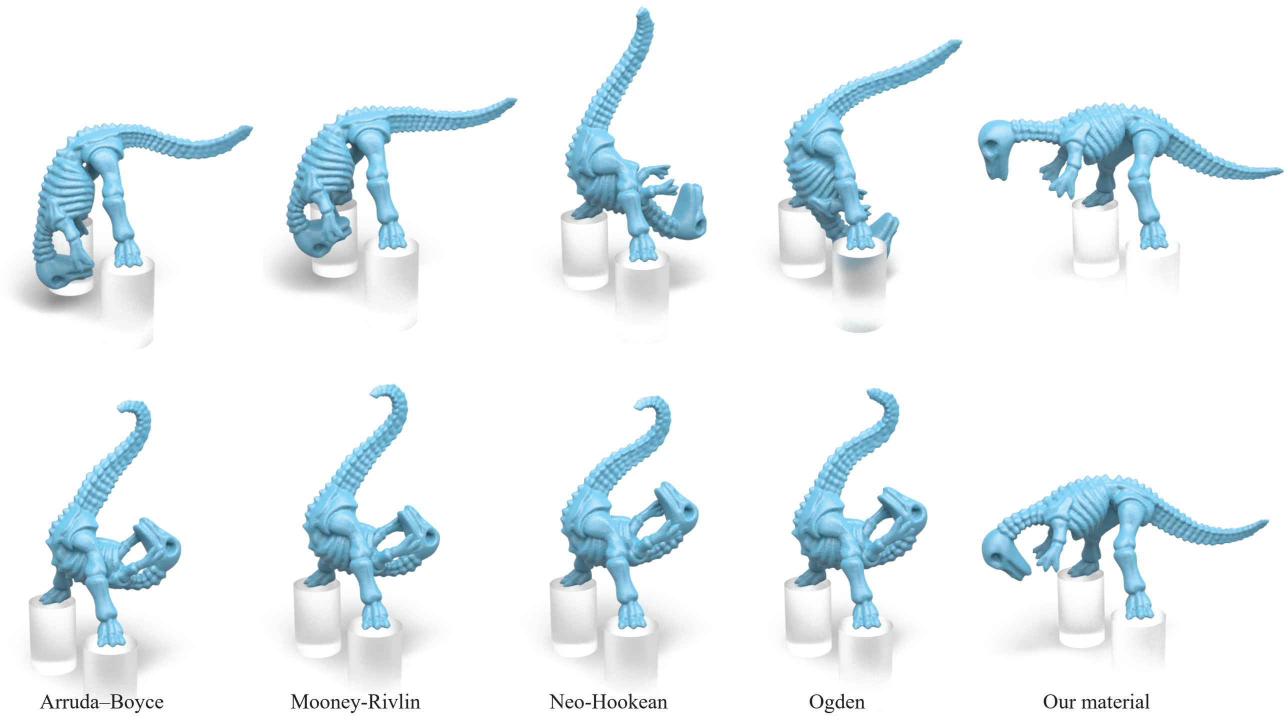

Fig. 16. MCFD allows us to compute higher-order modal derivatives that capture extreme bending effects of the dinosaur model (using 30 second-order derivative modes). Interestingly, the hyperelastic energy of Eq. (38), because of its strong resistance to volume change, cannot be bent as hard as other materials under the same external force.

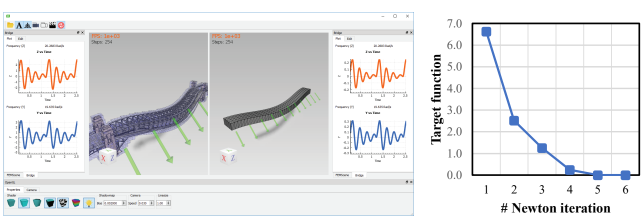

Fig. 17. We develop a system with an intuitive interface for the linear frequency design (left). The error reduces quickly along Newton iterations, with the Hessian accurately computed from MCFD. (right)

We show how this numerical algorithm can be applied for physics-based deformable simulation. Indeed, we believe that this method could be useful in a variety of graphics problems.

Limitation and future work The limitation of this method may be it requires a dedicated implementation in order to achieve a good performance. When CR form is used, the computation quickly becomes prohibitive if one wants to evaluate higher-order derivatives for a tensor-valued function. However, if the efficiency is not the primary concern, one can implement CFD and MCFD quickly based on any existing complex arithmetic library. In the future, we would like to fully leverage this new method to attack other challenging computational problems. For instance, to perform the imaginary perturbation along the time domain to get a better time integration. It is also of our great interests to apply this method to machine learning and other similar problems where optimizing complicated functions is required.

REFERENCES

Mark J Ablowitz and Athanassios S Fokas. 2003. *Complex variables: introduction and applications*. Cambridge University Press.

Rafael Abreu, Zeming Su, Jochen Kamm, and Jinghuai Gao. 2018. On the accuracy of the Complex-Step-Finite-Difference method. *J. Comput. Appl. Math.* 340 (2018), 390–403.

Lars V Ahlfors. 1973. Complex Analysis. 1979.

W Kyle Anderson, James C Newman, David L Whitfield, and Eric J Nielsen. 2001. Sensitivity analysis for Navier-Stokes equations on unstructured meshes using complex variables. *AIAA journal* 39, 1 (2001), 56–63.

RL Bagley. 2006. On Fourier differentiation – a numerical tool for implicit functions. *International Journal of Applied Mathematics* 19, 3 (2006), 255.

David Baraff. 1989. Analytical methods for dynamic simulation of non-penetrating rigid bodies. In *ACM SIGGRAPH Computer Graphics*, Vol. 23. ACM, 223–232.

David Baraff. 1991. Coping with friction for non-penetrating rigid body simulation. *ACM SIGGRAPH computer graphics* 25, 4 (1991), 31–41.

David Baraff and Andrew Witkin. 1998. Large steps in cloth simulation. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*. ACM, 43–54.

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

Jernej Barbič, Funshing Sin, and Eitan Grinspun. 2012. Interactive editing of deformable simulations. *ACM Transactions on Graphics (TOG)* 31, 4 (2012), 70.

Jernej Barbič, Fun Shing Sin, and Daniel Schroeder. 2012. Vega FEM Library.

Atilim Gunes Baydin and Barak A Pearlmutter. 2014. Automatic differentiation of algorithms for machine learning. *arXiv preprint arXiv:1404.7456* (2014).

Bradley M Bell. 2012. CppAD: a package for C++ algorithmic differentiation. *Computational Infrastructure for Operations Research* 57 (2012), 10.

Michael Betancourt. 2018. A Geometric Theory of Higher-Order Automatic Differentiation. *arXiv preprint arXiv:1812.11592* (2018).

Javier Bonet and Richard D Wood. 1997. *Nonlinear continuum mechanics for finite element analysis*. Cambridge university press.

Patrick Brezillon, Jean-François Staub, Anne-Marie Perault-Staub, and Gérard Milhaud. 1981. Numerical estimation of the first order derivative: approximate evaluation of an optimal step. *Computers & Mathematics with Applications* 7, 4 (1981), 333–347.

Robert Bridson. 2015. *Fluid simulation for computer graphics*. AK Peters/CRC Press.

N Butuk and J-P Pemba. 2003. computing CHEMKIN sensitivities using complex variables. *Journal of engineering for gas turbines and power* 125, 3 (2003), 854–858.

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

Xiang Chen, Changxi Zheng, Weiwei Xu, and Kun Zhou. 2014. An asymptotic numerical method for inverse elastic shape design. *ACM Transactions on Graphics (TOG)* 33, 4 (2014), 95.

N Fleury, M Rausch Detraubenberg, and RM Yamaleev. 1993. Commutative extended complex numbers and connected trigonometry. *Journal of mathematical analysis and applications* 180, 2 (1993), 431–457.

Rony Goldenthal, David Harmon, Raanan Fattal, Michel Bercovier, and Eitan Grinspun. 2007. Efficient simulation of inextensible cloth. In *ACM Transactions on Graphics (TOG)*, Vol. 26. ACM, 49.

Andreas Griewank, David Juedes, and Jean Utke. 1996. Algorithm 755: ADOL-C: a package for the automatic differentiation of algorithms written in C/C++. *ACM Transactions on Mathematical Software (TOMS)* 22, 2 (1996), 131–167.

Andreas Griewank and Andrea Walther. 2008. *Evaluating derivatives: principles and techniques of algorithmic differentiation*. Vol. 105. Siam.

Eitan Grinspun, Anil N Hirani, Mathieu Desbrun, and Peter Schröder. 2003. Discrete shells. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*. Eurographics Association, 62–67.

Gael Guennebaud, Benoit Jacob, et al. 2014. Eigen: a c++ linear algebra library. *URL http://eigen. tuxfamily. org, Accessed* 22 (2014).

Brian Guenter. 2007. Efficient symbolic differentiation for graphics applications. In *ACM Transactions on Graphics (TOG)*, Vol. 26. ACM, 108.

Fabian Hahn, Sebastian Martin, Bernhard Thomaszewski, Robert Sumner, Stelian Coros, and Markus Gross. 2012. Rig-space physics. *ACM transactions on graphics (TOG)* 31, 4 (2012), 72.

Fabian Hahn, Bernhard Thomaszewski, Stelian Coros, Robert W Sumner, and Markus Gross. 2013. Efficient simulation of secondary motion in rig-space. In *Proceedings of the 12th ACM SIGGRAPH/eurographics symposium on computer animation*. ACM, 165–171.

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

Robin J Hogan. 2014. Fast reverse-mode automatic differentiation using expression templates in C++. *ACM Transactions on Mathematical Software (TOMS)* 40, 4 (2014), 26.

IEEE. 1985. IEEE standard for binary floating-point arithmetic. Institute of Electrical and Electronic Engineers.

Geoffrey Irving, Joseph Teran, and Ronald Fedkiw. 2004. Invertible finite elements for robust simulation of large deformation. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*. Eurographics Association, 131–140.

Sanghaun Kim, Junghyun Ryu, and Maenghyo Cho. 2011. Numerically generated tangent stiffness matrices using the complex variable derivative method for nonlinear structural analysis. *Computer Methods in Applied Mechanics and Engineering* 200, 1-4 (2011), 403–413.

Yuki Koyama, Kenshi Takayama, Nobuyuki Umetani, and Takeo Igarashi. 2012. Real-time example-based elastic deformation. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. Eurographics Association, 19–24.

K-L Lai and JL Crassidis. 2008. Extensions of the first and second complex-step derivative approximations. *J. Comput. Appl. Math.* 219, 1 (2008), 276–293.

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), 16.

Sonia Lebofsky. 2013. *Numerically Generated Tangent Stiffness Matrices for Geometrically Non-Linear Structures*. Ph.D. Dissertation.

Abraham Lee. 2013. ad: Fast, transparent first- and second-order automatic differentiation. http://pythonhosted.org/ad

Tiantian Liu, Adam W Bargteil, James F O'Brien, and Ladislav Kavan. 2013. Fast simulation of mass-spring systems. *ACM Transactions on Graphics (TOG)* 32, 6 (2013), 214.

Tiantian Liu, Sofien Bouaziz, and Ladislav Kavan. 2017. Quasi-newton methods for real-time simulation of hyperelastic materials. *ACM Transactions on Graphics (TOG)* 36, 4 (2017), 116a.

JN Lyness. 1968. Differentiation formulas for analytic functions. *Math. Comp.* (1968), 352–362.

James N Lyness. 1967. Numerical algorithms based on the theory of complex variable. In *Proceedings of the 1967 22nd national conference*. ACM, 125–133.

V Maple. 1994. Waterloo maple software. *University of Waterloo, Version* 5 (1994).

Charles C Margossian. 2018. A Review of automatic differentiation and its efficient implementation. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* (2018), e1305.

Sebastian Martin, Bernhard Thomaszewski, Eitan Grinspun, and Markus Gross. 2011. Example-based elastic materials. In *ACM Transactions on Graphics (TOG)*, Vol. 30. ACM, 72.

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.

Don Mitchell and Pat Hanrahan. 1992. Illumination from curved reflectors. In *ACM SIGGRAPH Computer Graphics*, Vol. 26. ACM, 283–291.

Arturo Montoya, Randal Fielder, Armando Gomez-Farias, and Harry Millwater. 2014. Finite-element sensitivity for plasticity using complex variable methods. *Journal of Engineering Mechanics* 141, 2 (2014), 04014118.

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

Richard D Neidinger. 2010. Introduction to automatic differentiation and MATLAB object-oriented programming. *SIAM review* 52, 3 (2010), 545–563.

Jorge Nocedal and Stephen Wright. 2006. *Numerical optimization*. Springer Science & Business Media.

Alex Pentland and John Williams. 1989. *Good vibrations: Modal dynamics for graphics and animation*. Vol. 23. ACM.

Agustí Pérez-Foguet, Antonio Rodríguez-Ferran, and Antonio Huerta. 2000. Numerical differentiation for local and global tangent operators in computational plasticity. *Computer Methods in Applied Mechanics and Engineering* 189, 1 (2000), 277–296.

Griffith Baley Price. 1991. *An introduction to multicomplex spaces and functions*. M. Dekker.

Louis B Rall. 1981. Automatic differentiation: Techniques and applications. (1981).

Michael Renardy and Robert C Rogers. 2006. *An introduction to partial differential equations*. Vol. 13. Springer Science & Business Media.

Boris Schäling. 2011. *The boost C++ libraries*. Boris Schäling.

Adriana Schulz, Jie Xu, Bo Zhu, Changxi Zheng, Eitan Grinspun, and Wojciech Matusik. 2017. Interactive design space exploration and optimization for cad models. *ACM Transactions on Graphics (TOG)* 36, 4 (2017), 157.

Ken Shoemake. 1985. Animating rotation with quaternion curves. In *ACM SIGGRAPH computer graphics*, Vol. 19. ACM, 245–254.

Eftychios Sifakis and Jernej Barbic. 2012. FEM simulation of 3D deformable solids: a practitioner's guide to theory, discretization and model reduction. In *ACM SIGGRAPH 2012 Courses*. ACM, 20.

Breannan Smith, Fernando De Goes, and Theodore Kim. 2018. Stable Neo-Hookean Flesh Simulation. *ACM Transactions on Graphics (TOG)* 37, 2 (2018), 12.

William Squire and George Trapp. 1998. Using complex variables to estimate derivatives of real functions. *SIAM review* 40, 1 (1998), 110–112.

Ari Stern and Mathieu Desbrun. 2006. Discrete geometric mechanics for variational time integrators. In *ACM SIGGRAPH 2006 Courses*. ACM, 75–80.

Demetri Terzopoulos, John Platt, Alan Barr, and Kurt Fleischer. 1987. Elastically deformable models. *ACM Siggraph Computer Graphics* 21, 4 (1987), 205–214.

Christoph W Ueberhuber. 2012. *Numerical computation 1: methods, software, and analysis*. Springer Science & Business Media.

Andrew Voorhees, Harry Millwater, and Ronald Bagley. 2011. Complex variable methods for shape sensitivity of finite element models. *Finite elements in analysis and design* 47, 10 (2011), 1146–1156.

Huamin Wang and Yin Yang. 2016. Descent methods for elastic body simulation on the GPU. *ACM Transactions on Graphics (TOG)* 35, 6 (2016), 212.

Andrew Witkin. 1997. Physically Based Modeling: Principles and Practice Particle System Dynamics. *SIGGRAPH Course notes* (1997).

Stephen Wolfram et al. 1996. *Mathematica*. Cambridge university press Cambridge.

Hongyi Xu, Funshing Sin, Yufeng Zhu, and Jernej Barbič. 2015. Nonlinear material design using principal stretches. *ACM Transactions on Graphics (TOG)* 34, 4 (2015), 75.

Guowei Yan, Wei Li, Ruigang Yang, and Huamin Wang. 2018. Inexact descent methods for elastic parameter optimization. In *SIGGRAPH Asia 2018 Technical Papers*. ACM, 253.

Yin Yang, Dingzeyu Li, Weiwei Xu, Yuan Tian, and Changxi Zheng. 2015. Expediting precomputation for reduced deformable simulation. *ACM Transactions on graphics (TOG)* 34, 6 (2015).

A ELEMENTARY COMPLEX PROMOTION

The addition/subtraction and multiplication are trivial:

$$\begin{aligned} f(x_0) &= x_0 \pm a &\rightarrow\quad f^*(x_0 + hi) &= x_0 \pm a + hi, \\ f(x_0) &= s \cdot x_0 &\rightarrow\quad f^*(x_0 + hi) &= sx_0 + shi. \end{aligned} \tag{41}$$

The division is treated as the multiplication of the conjugate:

$$f(x_0) = \frac{a}{x} \rightarrow f^*(x_0 + hi) = \frac{a}{r^2}(x_0 - hi),\ r = \sqrt{x_0^2 + h^2}. \tag{42}$$

If the exponent of the power function (x^a) is an integer i.e. $a = n \in \mathbb{Z}$, we can use the De Moivre's formula:

$$f(x_0) = x^n \rightarrow f^*(x_0 + hi) = r^n(\cos n\phi + \sin n\phi i), \tag{43}$$

where $r\cos\phi = x_0$ and $r\sin\phi = h$ is the polar form of $x_0 + hi$. On the other hand, $a = 1/m$ ($m \in \mathbb{Z}$) makes $f(x_0)$ an m-root function, and the promotion is:

$$f(x_0) = x_0^{\frac{1}{m}} \rightarrow f^*(x_0 + hi) = r^{\frac{1}{m}} \left(\cos\frac{\phi + 2\pi k}{m} + \sin\frac{\phi + 2\pi k}{m} i\right). \tag{44}$$

Here, k is an integer between 0 and $m - 1$. In more general cases, when $a \in \mathbb{Q}$ is a rational number such that $a = n/m$, the power function of x^a is split as $f(x) = y^n$ and $y = a^{1/m}$.

The exponential function is promoted based on Euler's formula:

$$f(x_0) = e^{x_0} \rightarrow f^*(x_0 + hi) = e^{x_0}(\cos h + \sin hi). \tag{45}$$

The logarithmic promotion is the inverse of the exponential map, which can be obtained as:

$$f(x_0) = \ln x_0 \rightarrow f^*(x_0 + hi) = \ln r + (\phi + 2\pi k)i,\ k \in \mathbb{Z}. \tag{46}$$

Trigonometric functions can also be defined with complex numbers. According to Euler's formula, we have $\sin\alpha = (e^{\alpha i} - e^{-\alpha i})/2i$. Substituting α with $x_0 + hi$ leads to the promotion of $\sin x$:

$$f(x_0) = \sin x_0 \rightarrow f^*(x_0 + hi) = \frac{e^h + e^{-h}}{2}\sin x_0 + \frac{e^h - e^{-h}}{2}\cos x_0 i. \tag{47}$$

Similarly, $\cos\alpha = (e^{\alpha i} + e^{-\alpha i})/2$ is promoted as:

$$f(x_0) = \cos x_0 \rightarrow f^*(x_0 + hi) = \frac{e^h + e^{-h}}{2}\cos x_0 - \frac{e^h - e^{-h}}{2}\sin x_0 i. \tag{48}$$

Note that the promotion of exponential and logarithmic functions of Eqs. (44) and (46) is not unique due to the periodicity. We can restrict the argument angle to $[0, 2\pi]$, that makes $k = 0$.