Novel first and second order numerical differentiation techniques and their application to nonlinear analysis of Kirchhoff–Love shells

Leonardo Leonetti1,2 · Marat S. Mukhametzhanov2

Received: 6 July 2021 / Accepted: 10 February 2022 / Published online: 11 March 2022 © The Author(s) 2022

Abstract
Numerical simulation based on FEM/IGA methods is the standard approach for the approximated solution of applied physical problems. In this context, the differentiation of the numerical counterpart of mechanical fields is required. Moreover, the differentiated function can have a complicated shape, depend on many variables and change within the process. Many state-of-the-art numerical differentiation methods are not suitable for this kind of applications and the common way is to exploit analytical differentiation. Thus, an on-the-fly differentiation method is desirable particularly when the process is complicated and when new mechanical models are under development. In this paper, a new method is proposed for a precise computation of the gradient and Hessian. This method has been applied to nonlinear analysis of Kirchhoff–Love shells, which can be considered as an appropriate test bench to prove the reliability in relevant physical context. Numerical experiments show the advantages of the proposed techniques with respect to standard approaches.

Keywords Geometric nonlinearities · Isogeometric analysis · Kirchhoff–Love shells · Infinity computer · Numerical differentiation · Numerical infinitesimals

1 Introduction

Finite Element Analysis is the standard approach for solving a large number of problems based on partial differential equations. It is typical for applied mathematics and physics due to its simplicity in software implementation on modern computers, which become nowadays faster and faster. Recently Isogeometric analysis (IGA) (see, e.g., [9,42]) has been proposed as an alternative discretization tool allowing one, in connection with CAD, to exactly describe the geometry. IGA is based on interpolation spline functions (like, e.g., B-splines or NURBS), which make it possible to fulfil the continuity requirement easily, particularly suitable for problems which require high order derivatives.

Isogeometric analysis has attracted the attention of many researchers in the context of different applications based on various mechanical models (see, e.g., [27,28,30,33,36,37]). Moreover, interesting industrial applications have been presented recently for the optimization of post-buckling behaviour of aeronautical specimens made of advanced composite materials in [19,34]. In the following we refer to the application of these numerical techniques to structural mechanical problems. Typically, for the numerical solution of mechanical problems, it is required to evaluate the variation of the potential energy which leads to variations of strain measure addressed in the paper. The derivation of the stress with respect to strain is usually required for the linearization in terms of the Newton–Raphson method. Depending on the chosen mechanical model for the analysis, this derivation procedure may be complex and thus cumbersome to compute and implement, sometimes due to their analytical formulae the derivation of the stress with respect to strain.

In particular, the Kirchhoff–Love model is potentially able to furnish the most efficient analysis of shell structures, due to the reduced number of degrees of freedom (DOF) paying for this advantage a more complicated strain measure, compared to solid-shell models (see, e.g., [28]), and a consequent effort for the analytical derivation and the computation
of the discrete quantities, even though, in this respect, more complex applications in multi-physics analysis can be found. The aim of this work is to propose a numerical differentiation tool that is less error prone, for the construction of the numerical problem. Moreover, engineering models of appreciable complexity are typically constructed using multiple surface patches. An efficient proposal for the patch-coupling has been presented in [29] that makes efficient the application to large scale problems in which extra non trivial terms are introduced and make the analytical derivation process much more difficult. For these reasons, the Kirchhoff–Love model is a good candidate to show the effectiveness of the derivation procedure proposed in the present paper.

As it has been already mentioned earlier, the problem of the efficient computation of the first- and second-order derivatives with respect to the strain measure arises. There are different ways to calculate the derivatives with a high accuracy: Automatic differentiation (see, e.g., [4,18,55]), symbolic differentiation (see, e.g., MATLAB symbolic toolbox https://mathworks.com/products/symbolic.html or open source SymPy library for Python https://www.sympy.org/); differentiation using frameworks for working with numerical infinitesimals (see, e.g., [47,52]); differentiation using complex or dual numbers (see, e.g., [26,54,56]), etc. Moreover, there has been proposed a multi-language and multi-environment software tool (called AceGen, see [23,24]) for the automatic code generation using Wolfram Mathematica system as the core. In particular, this tool allows to generate the formulae for the required derivatives by combining Mathematica’s symbolic and algebraic procedures with automatic differentiation and code generation, simultaneous optimization of expressions and theorem-proving via stochastic evaluation of expressions. For these reasons, AceGen can also be successfully used in FEM based analysis.

However, some properties of the function to be differentiated create several difficulties and limitations for these methods. In particular, since the strain measure can be multidimensional with, eventually, high number of variables, then it is required to calculate partial and mixed second-order derivatives for each evaluation point. Second, the objective function depends on additional data and parameters (e.g., current displacements and tensions), which are changed or obtained during the simulation. This leads to the fact that different evaluations of the gradient and Hessian are executed on different functions, which makes it difficult to apply differentiation techniques requiring a pre-processing of the software (e.g., algorithmic (or automatic) differentiation, which requires the construction of the code for the derivatives, or symbolic differentiation, which constructs the formulae for the respective derivatives). For this issue, only differentiation techniques, which do not require an explicit generation of the code or formulae of the derivatives, like the ones using numerical infinitesimals or complex step methods, can be applied for this purpose. Moreover, AceGen described above has also several limitations in this context. First of all, even if it is multi-language and multi-environment, it requires the Mathematica system, which is not currently available for free, while the approach proposed in the present paper is domain-independent, general-purpose and can be easily implemented in any language and environment using available frameworks for working numerically with infinite and infinitesimal numbers. Second, in order to apply this tool, the finite element formulation must be developed using the same programming environment and language (however, not necessarily Mathematica) by developing all the manipulations required by the chosen model starting from the expressions of shape functions, that are usually simple polynomial formulae in case of standard FEMs. It is a bit different in case of NURBS based analysis, for which the shape functions and their derivatives are evaluated at a point by using procedures [32], while their analytical expression is, often, not explicitly given. However, it should be noted that AceGen can also be used in IGA analysis (see, e.g., the AceFEM package http://symech.fgg.uni-lj.si/ and https://www.wolfram.com/products/applications/acefem/) representing so an alternative available software solution in this context.

There exist several differentiation techniques using numerical infinitesimals in the literature. In [47], there has been proposed the method for differentiation of a univariate function. However, it has been constructed for univariate derivatives only and has not been implemented for the gradient or Hessian. Multidimensional differentiation techniques have been also proposed in [20,21], but they are applicable only for Lie derivatives embedded in initial value problems. In [52], there has been proposed the method for higher-order mixed differentiation. However, the method from [52] requires a numerical solution to ill-conditioned linear systems (or inversion of the respective matrix), the dimension of which grows significantly, if the objective function has many variables and dimensions. In particular, it requires to solve the \( l \times l \) ill-conditioned linear system for finding the Hessian, where \( l = \frac{m(m+1)}{2} \) and \( m \) is the number of variables, leading so to both higher computational costs (e.g., inverting and multiplying matrices or solving linear systems of a high dimension) and lower accuracy (since the respective linear system can become ill-conditioned). To overcome the issues described above, a new method for computation of both the gradient and Hessian is proposed in this paper. This method is suitable for the frameworks for working with infinite and infinitesimal numbers. It is shown that the proposed method is simple, efficient and easily parallelizable, allowing one to calculate the required derivatives on-the-fly with a high precision and not involving ill-conditioned linear systems even if the objective function is changing within an iterative process. Numerical simulation results show the advantages of the proposed method with its direct competitors.
Proposed differentiation method is based on the efficient computation of the derivatives for a univariate function. In this paper, the frameworks for working numerically with infinite and infinitesimal quantities: see, e.g., [48] for the Infinity Computer, [52] for the Levi-Civita fields, [44] for the Non-Standard Analysis, [5] for labeled sets and numerosities, etc. In this paper, the Infinity Computer (see, e.g., [46,48] for its complete description and the patents [45] for technical implementation details) is used for this purpose, since it has already been successfully used in different applications, where a high precision computation of the derivatives is involved (see, e.g., [13,20,21,47,51]). Moreover, it has been also successfully used in many other different applied fields, e.g., in optimization [7,8,11,12], in ordinary differential equations [2,50], in handling ill-conditioning and high precision computations [1,17,49], numerical simulation and modeling [13,14], etc. However, it should be noted that the method proposed and described in this paper does not depend explicitly on the structure of the objective function and its derivatives, sub- stantiating so the necessity of an efficient differentiation method.

The rest of the paper is organized as follows. Section 2 briefly describes the Kirchhoff–Love shell model. Section 3 presents the differentiation techniques and briefly describes the chosen framework for working with infinite and infinitesimal numbers. Sect. 4 presents the results of the numerical comparison of the proposed methods with their direct competitors on several benchmark test problems. Section 5 presents the numerical experiments and obtained results. Section 6 concludes the paper. Finally, analytical strain variations are presented in Appendix in order to show the complex structure of the objective function and its derivatives, substantiating so the necessity of an efficient differentiation method.

2 Kirchhoff–Love shells in large deformation

Kirchhoff–Love theory assumes that transverse shear strains are negligible, for this reason the model well adapts thin shells widely used in industrial applications. In this case, the shell kinematics is described in terms of displacement variables only and second order derivatives appear in the governing variational equations of the Kirchhoff–Love theory. This implies the necessity of $C^1$-continuous approximation functions, which has always been a major obstacle for the development of efficient finite element formulations for thin shells, which have been overcome by using Spline based interpolations [3,22].

2.1 Kirchhoff–Love shell kinematics

The model described in this section has been presented in [22], to which the reader is invited to refer for a deeper discussion. Herein, we use a Total Lagrangian formulation to identify material points on the middle surface of the current configuration in terms of their position vector $X(\xi, \eta)$ in the reference configuration and the displacement field $u(\xi, \eta)$, as follows

$$x(\xi, \eta) = X(\xi, \eta) + u(\xi, \eta),$$

where $\xi = [\xi, \eta]$ denotes convective curvilinear shell coordinates with $(\xi, \eta)$ representing in-plane coordinates. The middle surface covariant basis vectors in the undeformed and actual configuration are by definition, respectively

$$G_i = X_{,i}, \quad g_i = x_{,i} = G_i + u_{,i} \quad \text{with} \quad i = 1, 2,$$

where the symbol $(,)_i$ is used to denote the partial derivation with respect to the $i$-th component of $\xi$, while the unit normal ones are

$$G_3 = G_1 \times G_2, \quad A_3 = \frac{G_3}{\|G_3\|},$$

$$g_3 = g_1 \times g_2, \quad a_3 = \frac{g_3}{\|g_3\|}.$$  

This is sufficient to impose the Kirchhoff–Love (KL) shell constraint: the director remains straight and normal to the mid-surface during deformation and the Green–Lagrange strain tensor reduces to

$$E = \bar{E}_{ij} G^i \otimes G^j \quad i, j = 1, 2,$$  

where $\bar{E}_{ij}$ are the covariant strain components. Assuming the strain to vary linearly through the thickness, it is splitted into a membrane (constant) a bending (linear) part respectively. The covariant strain coefficients are

$$\bar{E}_{ij} = \bar{e}_{ij} + \zeta \bar{\kappa}_{ij} = \frac{1}{2} (g_{ij} - G_{ij}) + \zeta (B_{ij} - b_{ij})$$

with $i, j = 1, 2,$

$$\bar{e}_{ij} = \bar{e}_{ij},$$

with $\zeta \in [-t/2, t/2]$ and $t$ being the thickness of the shell. The curvature tensor coefficients are defined as $[22]$

$$B_{ij} = -\frac{1}{2} (G_{i} \cdot A_{3,j} + G_{j} \cdot A_{3,i}) = G_{i,j} \cdot a_{3},$$

$$b_{ij} = -\frac{1}{2} (g_{i} \cdot a_{3,j} + g_{j} \cdot a_{3,i}) = g_{i,j} \cdot a_{3}.$$
Following the isoparametric concept, geometry and displacement fields are approximated, over the element, as follows

\[
X(\xi, \eta) = N_u(\xi, \eta)X_e, \quad u(\xi, \eta) = N_u(\xi, \eta)d_e,
\]

where \( X_e \) and \( d_e \) collect the element control points and the element control displacements, respectively. The matrix \( N_u(\xi, \eta) \) collects bivariate NURBS functions [9]. Adopting Voigt’s notation, the covariant strain components in (5) are collected in the vector \( \bar{\varepsilon} = [\bar{\varepsilon}_{\xi\xi}, \bar{\varepsilon}_{\eta\eta}, 2\bar{\varepsilon}_{\xi\eta}]^T \), that, exploiting (6), becomes

\[
\bar{\varepsilon} = \bar{\varepsilon} + \zeta \bar{\chi},
\]

with \( \bar{\varepsilon} = [\bar{\varepsilon}_{\xi\xi}, \bar{\varepsilon}_{\eta\eta}, 2\bar{\varepsilon}_{\xi\eta}]^T \) and \( \bar{\chi} = [\bar{\chi}_{\xi\xi}, \bar{\chi}_{\eta\eta}, 2\bar{\chi}_{\xi\eta}]^T \). The generalized covariant strain vector \( \tilde{\varepsilon} \equiv [\bar{\varepsilon}, \bar{\chi}]^T \) by exploiting the isogeometric approximation, \( \check{\varepsilon} \) becomes

\[
\check{\varepsilon} = \check{D}(\xi, \eta, d_e)d_e,
\]

where \( \check{D} \) depends on the displacement DOFs. The map between parametric and physical domain [31] allows to obtain their Cartesian expression as

\[
e = D(\xi, \eta, d_e)d_e.
\]

The mapping between the parametric and the physical domains and the homogenized constitutive matrix \( C_e \) necessary to construct coherently the mechanical model are given in [31] and \( \sigma = C_e \varepsilon \).

### 2.2 Nonlinear analysis framework

The equilibrium of slender elastic structures subject to conservative loads \( f \) amplified by a proportionality factor \( \lambda \) is expressed by the virtual work equation

\[
\Phi(u')\delta u - \lambda f \delta u = 0, \quad u \in \set{U}, \quad \delta u \in \set{T},
\]

where \( u \in \set{U} \) is the field of configuration variables, \( \Phi(u) \) denotes the strain energy, \( \set{T} \) is the tangent space of \( \set{U} \) at \( u \) and \( ()' \) is used to express the Fréchet derivative with respect to \( u \). \( \set{U} \) is a linear manifold so that its tangent space \( \set{T} \) is independent of \( u \). The discrete counterpart of (10) is

\[
r(d, \lambda) = s(d) - \lambda f = 0, \quad \text{with} \quad \begin{cases} 
    s^T \delta d & \equiv \Phi'(u)\delta u \\
    f^T \delta d & \equiv f \delta u,
\end{cases}
\]

where \( r: \mathbb{R}^{N+1} \rightarrow \mathbb{R}^N \) is a nonlinear vectorial function of the vector \( z \equiv \{ d, \lambda \} \in \mathbb{R}^{N+1} \), collecting the configuration \( d \in \mathbb{R}^N \) and the load multiplier \( \lambda \in \mathbb{R} \), \( s(d) \) is the internal force vector, and \( f \) the reference load vector. (11) represents a system of \( N \) equations and \( N+1 \) unknowns and its solutions define the \textit{equilibrium paths} as curves in \( \mathbb{R}^{N+1} \). The Riks’ approach [43] can be used to trace these curves step-by-step from a known initial configuration \( d_0 \) corresponding to \( \lambda = 0 \). At each step some Newton iterations are needed to solve (11). To this end, we also define the tangent stiffness matrix as

\[
\delta d^T K(d) \delta d = \Phi''(u)\delta u \delta u, \quad \forall \delta d, \delta d,
\]

where \( \delta u \) and \( \tilde{u} \) are generic variations of the configuration field \( u \) and \( \delta d \) and \( \tilde{d} \) the corresponding discrete vectors.

In displacement-based formulations, the strain energy can be expressed as a sum of element contributions \( \Phi_e(u) = \sum_e \Phi_e(u) \)

\[
\Phi_e(u) = \int_{\Omega_e} \left( \frac{1}{2} \varepsilon^T \sigma \right) \, d \Omega_e,
\]

where \( \Omega_e \) is the element domain. The first variation of the generalized strains in (9) can be written as

\[
\delta \varepsilon = B(d_e) \delta d_e,
\]

and, then, the first variation of the strain energy is

\[
\Phi_e(u)\delta u = \int_{\Omega_e} \left( \delta \varepsilon^T \sigma \right) \, d \Omega_e
= \delta d_e^T \int_{\Omega_e} \left( B(d_e)^T \sigma \right) \, d \Omega_e
= \delta d_e^T \sigma_s(d_e),
\]

where \( \sigma_s(d_e) \) is the element internal force vector. The second variation of the strain measure is

\[
\delta \tilde{\varepsilon} = Q(d_e, \tilde{d}_e) \delta d_e = Q(d_e, \delta d_e) \tilde{d}_e,
\]

and its \( k \)-th component can be evaluated, introducing the symmetric matrix \( \Psi_k(d_e) \), as

\[
\delta \tilde{\varepsilon}_k = \tilde{d}_e^T \Psi_k(d_e) \delta d_e.
\]

Letting \( \sigma(d_e) = C_e \varepsilon(d_e) \), the following expression holds

\[
\sigma^T \delta \tilde{\varepsilon} \equiv \sum_k \sigma_k \delta \tilde{\varepsilon}_k = \tilde{d}_e^T \mathcal{G}(d_e, \sigma(d_e)) \delta d_e,
\]

with

\[
\mathcal{G}(d_e, \sigma(d_e)) = \sum_k \sigma_k \Psi_k(d_e).
\]

The second variation of the strain energy is

\[
\Phi_e''(u)\delta u \tilde{u} = \int_{\Omega_e} \left( \delta \varepsilon^T \sigma + \tilde{\varepsilon}^T \sigma \right) \, d \Omega_e
\]
\[ \delta \mathbf{d}_e^T \mathbf{K}_e(\mathbf{d}_e) \delta \mathbf{d}_e, \]  
\[ \text{with the element tangent stiffness matrix defined as} \]
\[ \mathbf{K}_e(\mathbf{d}_e) \equiv \int_{\Omega_e} \left( \mathbf{B}(\mathbf{d}_e)^T \mathbf{C}_e \mathbf{B}(\mathbf{d}_e) + \mathbf{G}(\mathbf{d}_e, \sigma(\mathbf{d}_e)) \right) \, d\Omega_e. \]  

The discrete operators involved in the evaluation of the internal force vector and stiffness matrix are defined by means of the strain derivatives. In order to avoid locking, the patch-wise reduced integration proposed in [31] is used to evaluate the integrals in (15) and (19). The proposed derivation technique is used to evaluate the derivatives \( \delta \mathbf{e} \) and \( \delta \mathbf{e}^T \) of the discrete strain measure \( \mathbf{e} \) with respect to the input vector \( \mathbf{d} \). As it is shown in Appendix, the formulae for the analytical evaluation of \( \delta \mathbf{e} \) and \( \delta \mathbf{e}^T \) using (14) and (16) can be complicated as well as their explicit generation. Moreover, these functions can depend not only on the input \( \mathbf{d} \) but on the additional parameters and structures changing during the execution. For these reasons, an efficient tool for computation of these derivatives is required.

The isogeometric shell model described in this section with the patch-wise reduced integration rule proposed in [31] is accurate and efficiently solved exploiting the so-called Mixed Integration Point strategy [16,35], proposed in order to overcome the inefficiency of standard displacement-based finite element problems and then extended and tested in displacement-based isogeometric formulations [28,38,39].

### 3 Differentiation techniques

#### 3.1 General considerations

The following Proposition reformulates the multidimensional differentiation problem of computation of the gradient and Hessian to the computation of the respective univariate derivatives.

**Proposition 1** Let \( f(\mathbf{x}) \) be a scalar-valued function, at least twice continuously differentiable at \( \mathbf{x}^0 = (x_1^0, x_2^0, \ldots, x_m^0) \). Let also \( y_i(t) \) and \( y_{ij}(t) \), \( i, j = 1, \ldots, m \), be univariate functions defined as follows:

\[ y_i(t) = f(\mathbf{x}^0 + t \mathbf{e}_i), \]  
\[ y_{ij}(t) = f(\mathbf{x}^0 + t (\mathbf{e}_i + \mathbf{e}_j)), \]

where \( \mathbf{e}_i \) is the \( i \)-th column of the \( m \)-dimensional identity matrix, \( t, t \in \mathbb{R} \), is an auxiliary univariate variable. Then, the following equations are valid:

\[ \nabla_i f(\mathbf{x}^0) = y'_i(0), \]  
\[ H_{ii}(\mathbf{x}^0) = y''_i(0), \]  
\[ H_{ij}(\mathbf{x}^0) = \frac{1}{2} \left[ y''_{ij}(0) - H_{ii}(\mathbf{x}^0) - H_{jj}(\mathbf{x}^0) \right], \]  

where \( \nabla_i f(\mathbf{x}^0), H_{ii}(\mathbf{x}^0), \) and \( H_{ij}(\mathbf{x}^0) \) are the \( i \)-th component of the gradient \( \nabla f(\mathbf{x}^0) \), \( i \)-th diagonal entry and \( (i, j) \)-th non-diagonal entry of the Hessian matrix \( \mathbf{H}(\mathbf{x}^0) \).

**Proof** In order to prove this Proposition, let us consider the following Taylor expansion of \( f(\mathbf{x}) \) around \( \mathbf{x}^0 \):

\[ f(\mathbf{x}) = f(\mathbf{x}^0) + (\mathbf{x} - \mathbf{x}^0)^T \nabla f(\mathbf{x}^0) \]
\[ + \frac{1}{2!} (\mathbf{x} - \mathbf{x}^0)^T \mathbf{H}(\mathbf{x}^0) (\mathbf{x} - \mathbf{x}^0) \]
\[ = f(\mathbf{x}^0) + \sum_{i=1}^{m} f_{,i}(\mathbf{x}^0)(x_i - x_i^0) \]
\[ + \frac{1}{2!} \sum_{i=1}^{m} \sum_{j=1}^{m} f_{,ij}(\mathbf{x}^0)(x_i - x_i^0)(x_j - x_j^0) + \ldots \]  
\[ (25) \]

On the one hand, substituting \( \mathbf{x} = \mathbf{x}^0 + t \mathbf{e}_i \) in (25), one can obtain that

\[ y_i(t) = f(\mathbf{x}^0 + t \mathbf{e}_i) = f(\mathbf{x}^0) + f_{,i}(\mathbf{x}^0)t \]
\[ + \frac{1}{2!} f_{,ii}(\mathbf{x}^0)t^2 + \ldots, \]  
\[ (26) \]

being the Taylor expansion of \( y_i(t) \) around \( t = 0 \). It should be noted that \( \mathbf{x} = \mathbf{x}^0 + t \mathbf{e}_i \) differs from \( \mathbf{x}^0 \) at only one \( i \)-th component \( x_i^0 \), while all the remaining components coincide. Thus, the first- and second-order derivatives \( f_{,i}(\mathbf{x}^0) \) and \( f_{,ii}(\mathbf{x}^0) \) are equal to \( y'_i(0) \) and \( y''_i(0) \), respectively, i.e., Eqs. (22)–(23) hold.

On the other hand, substituting \( \mathbf{x} = \mathbf{x}^0 + t(\mathbf{e}_i + \mathbf{e}_j) \) in (25), one can also obtain that

\[ y_{ij}(t) = f(\mathbf{x}^0 + t(\mathbf{e}_i + \mathbf{e}_j)) \]
\[ = f(\mathbf{x}^0) + [f_{,i}(\mathbf{x}^0) + f_{,j}(\mathbf{x}^0)]t \]
\[ + \frac{1}{2!} [f_{,ii}(\mathbf{x}^0) + 2f_{,ij}(\mathbf{x}^0) + f_{,jj}(\mathbf{x}^0)]t^2 + \ldots, \]  
\[ (27) \]

being the Taylor expansion of \( y_{ij}(t) \) around \( t = 0 \). Thus, the second-order derivative \( y''_{ij}(0) \) is equal to \( f_{,ii}(\mathbf{x}^0) + 2f_{,ij}(\mathbf{x}^0) + f_{,jj}(\mathbf{x}^0) \), from where it is easy to see that (24) is also satisfied. \( \square \)

One can see from (22), (23), and (24) that both the gradient and Hessian can be successfully calculated using differentiation techniques for a univariate function. It is shown in the following sections how to apply these techniques for the computation of the gradient and Hessian on the Infinity Computer—a framework for working numerically with infinite and infinitesimal numbers, which can be used for computation of the respective univariate derivatives.
3.2 Differentiation on the infinity computer

The Infinity Computer is based on using of the positional numeral system with the infinite radix $\hat{\Xi}$ (called grossone, see, e.g., [45,48] for details). A number $X$ in this framework can be represented as follows:

$$X = c_0\hat{\Xi}^{p_0} + c_1\hat{\Xi}^{p_1} + c_2\hat{\Xi}^{p_2} + \cdots + c_N\hat{\Xi}^{p_N}, \quad (28)$$

where $c_i$, $i=0,\ldots,N$, are finite (positive or negative) floating-point numbers, while $p_i$, $i=0,\ldots,N$, can be finite, infinite, and infinitesimal of the form (28) and ordered in a decreasing order, $p_0 > p_1 > \cdots > p_N$. In this paper, a software simulator with only finite powers $p_i$ is used, since they are sufficient for the differentiation purposes. Thus, it is supposed that $p_i$, $i=0,\ldots,N$, are also finite floating-point numbers. The number $N$ is the maximum number of powers used in computations and represents the precision: results of arithmetic operations are truncated after $N$ powers of $\hat{\Xi}$ or, in other words, after $N$ items in (28).

In the Infinity Computing framework, the number $\hat{\Xi}$ possesses all properties of a natural number, including $\hat{\Xi}^0 = 1$. Thus, the item $c_i\hat{\Xi}^{p_i}$, where $p_i = 0$, represents the finite part of (28) (if there is no such an item in (28), then the finite part is equal to zero). The number $X$ having at least one finite positive power $p_i$ in (28) is called infinite, while the number having only negative finite powers is called infinitesimal. Finally, the number $X = c\hat{\Xi}^0$, i.e., having only the power of $\hat{\Xi}$ equal to zero, is called purely finite. A more detailed description of the Infinity Computer including its technical implementation information can be found, e.g., in [14,21,45].

In different papers, there have been already proposed several methods for computation of the first- and second-order derivatives of a purely finite univariate function $y(t)$ (i.e., a function not depending explicitly on $\hat{\Xi}$ and having only purely finite values at any purely finite $t$) at the purely finite point $z$ using numerical infinitesimals, for example, the Taylor-based method on the Infinity Computer proposed in [47] and extended in [13] for Simulink. This method is based on the Taylor expansion of $y(t)$:

$$y(t) = y(z) + y'(z)(t - z) + \frac{y''(z)}{2}(t - z)^2 + \frac{y'''(z)}{6}(t - z)^3 + \cdots, \quad (29)$$

As it was shown in [47], in order to find the derivatives of $y(t)$ at the point $z$ numerically, the value $y(z + h)$ is calculated, where $h$ is infinitesimal (the value $h = \hat{\Xi}^{-1}$ was used in [47], but this method is valid for other infinitesimals as well):

$$y(z + h) = c_0 + c_1h + c_2h^2 + c_3h^3 + \cdots, \quad (30)$$

where $c_k$, $k = 0, 1, 2, \ldots$, are purely finite numbers. It should be noted that the result in (30) is written down in powers of $h$ instead of the powers of $\hat{\Xi}$ as in (28) just for simplicity: all computations are performed in the form (28), while the coefficients of $h$ in (32) can be easily calculated from the coefficients of $\hat{\Xi}$ in (28). Moreover, if, e.g., $h = \hat{\Xi}^{-1}$ is chosen, then the coefficients of $h^i$, $i = 0, 1, \ldots, i$ in (32) are equal to the coefficients of $\hat{\Xi}^{-i}$, $i = 0, 1, \ldots, i$ in (28).

As it has been shown in [47], the finite part $c_0$ in (30) corresponds to the purely finite value $y(z)$. Moreover, the coefficients $c_k$, $k = 1, 2, \ldots$, give us the $k$-th derivatives of $y(t)$ at $z$ up to machine precision:

$$y^{(k)}(z) = k!c_k. \quad (31)$$

It should be noted that, since the Infinity Computer uses numerical and not symbolic infinitesimals, all computations are performed using floating-point arithmetic. For this reason, the error in the computation of the respective coefficients $c_k$ is equivalent to the machine precision.

The algorithms proposed in the following are described in terms of the different forms of the Infinity Computer framework. However, it is easy to see that these methods can also be used with any other similar framework, e.g., using Levi-Civita field ([52]) or hyper-dual numbers ([54]). However, one can easily see that the Infinity Computer with finite powers of $\hat{\Xi}$ (which is used in the present paper) can be simply considered as a particular implementation of the Levi-Civita field substituting the Levi-Civita’s generic infinitesimal $\varepsilon$ by a concrete $\hat{\Xi}$-based infinitesimal, e.g., $\hat{\Xi}^{-1}$ (but a detailed comparison of these two methodologies is out of scope for the present paper).

On the other hand, the concepts of the hyper-dual numbers allow to work only with finite and infinitesimal numbers without possibility of working with infinite numbers in the same framework. However, it has been shown in several papers that the possibility of working with infinite numbers can be very important from the practical point of view: e.g., in penalty functions and optimization (see, e.g., [10]), handling ill-conditioning ([17,49]), truncated characteristic polynomial for quadratic matrices ([41]), etc. For these reasons, a general-purpose framework like the Infinity Computer or Levi-Civita field is more appropriate for a complex analysis of real-life processes instead of hyper-dual numbers. Moreover, the hyper-dual numbers use different number of “axis” w.r.t. the proposed approach. The algorithms for the computation of the scalar-valued second-order derivatives on the Infinity Computer (31) use only 3 axis being the powers 0, 1, and 2 of $h$ in (30). The hyper-dual numbers, instead, use different infinitesimals $\varepsilon_1$ and $\varepsilon_2$, so each computation of the scalar-valued second-order derivatives requires 4 axis: the finite (real) part of the result, the “imaginary” parts (the powers 1 of $\varepsilon_1$ and $\varepsilon_2$, respectively, being the first order
derivatives) and the term $\varepsilon_1 \varepsilon_2$ representing the second-order derivative. So the evaluations used in the proposed approach are expected to be lighter than those required by the hyper-
dual numbers. However, their fair comparison requires an
efficient software implementation of the Infinity Computer,
which is absent at this moment and its realization requires
vast scientific and software development contributions (out
of scope of the current paper).

3.3 Computation of the gradient

Let us modify the method from [13,47] for computation of
the gradient $\nabla f(x^0)$ using (22). First, the function $y_i(t) = f(x^0 + t e^i)$ can be defined according to (20), where $e^i$ is
the $i$-th column of the $m$-dimensional identity matrix. Then,
its first-order derivative can be easily calculated at $t = 0$
using (30) as follows. The function $f$ is evaluated at
the point $x^0 + h e^i$, where $h$ is a numerical infinitesimal of
the form (28):

$$f(x^0 + h e^i) = a_0 + a_1 h + a_2 h^2 + \ldots,$$

(32)

where $a_k$, $k = 0, 1, \ldots$, are purely finite numbers.

As it has been shown previously, the finite part $a_0$ in (32)
corresponds to the purely finite value $f(x^0)$, while the
coefficient $a_1$ corresponds to the first-order derivative:

$$f_{,i}(x^0) = a_1 = (f(x^0 + h e^i) - f(x^0)) h^{-1} + O(h),$$

(33)

where the error in computation of $f_{,i}(x^0)$ is an infinitesimal
of order $O(h)$, so the finite part of $(f(x^0 + h e^i) - f(x^0)) h^{-1}$
gives us the required first partial derivative up to machine
precision. However, since for a purely finite $x^0$, the value
$f(x^0)$ is also purely finite, so $f(x^0) h^{-1}$ is infinitesimal
(or equal to zero if $f(x^0) = 0$). Thus, the finite part of
$(f(x^0 + h e^i) - f(x^0)) h^{-1}$ is simply equal to the finite part
of $(x^0 + h e^i) h^{-1}$, which gives us exactly the required
first-order partial derivative (being also the $i$-th component
$\nabla_i f(x^0)$ of the gradient $\nabla f(x^0)$):

$$\nabla_i f(x^0) = f_{,i}(x^0) = \text{FinitePart}(f(x^0 + h e^i) h^{-1}),$$

(34)

where $\text{FinitePart}[x]$ is the finite part of the number $x$, i.e.,
the coefficient of $\Xi_0$ in (28). In other words, the $i$-th partial
derivative $f_{,i}(x^0)$ (being the $i$-th component of $\nabla f(x^0)$) is
the coefficient of the infinitesimal $h$ in $f(x^0 + h e^i)$ given
in the form (28): if, e.g., $h = \Xi_0^{-1}$, then the $i$-th component
of the gradient can be calculated extracting the coefficient of
$\Xi_0^{-1}$ from $f(x^0 + \Xi_0^{-1} e^i)$. Formally, the algorithm for the
computation of the gradient is presented below:

Algorithm 1 Computation of the gradient of the purely finite
function $f(x)$, $f : \mathbb{R}^m \rightarrow \mathbb{R}$, at the purely finite point
$x^0 = (x_{10}, x_{20}, \ldots, x_{m0})$

1: Fix the infinitesimal $h$ in form (28).
2: for $i \leftarrow 1$ to $m$
3: \quad Calculate $f(x^0 + h e^i)$ according to (32),
4: \quad Extract the coefficient $a_1$ from (32) using, e.g., (34),
5: \quad Assign $\nabla_i f(x^0) \leftarrow a_1$.
6: end for
7: return $\nabla f(x^0)$

One can see that the Algorithm 1 is simple and easily
parallelizable along the main “for” loop in case of large
dimensions $m$. An example of the application of the Algo-
rithm 1 is presented below.

Example 1 Let us calculate the gradient of the function
$$f(x) = \frac{x_1^2 + x_2^2}{x_1 + x_2 - 2x_3}$$

at $x^0 = (1, 2, 0)$. First, let us fix the
infinitesimal $h$, e.g., equal to the simplest infinitesimal of
the form (28): $h = \Xi_0^{-1}$. Thus, $\nabla f(x^0) = (f_{,1}(x^0), f_{,2}(x^0))$.

Let us first calculate $f(x^0 + h e^1)$:

$$f(x^0 + h e^1) = f(1 + \Xi_0^{-1}, 2)$$

$$= ((1 + \Xi_0^{-1})^2 + 2^2)/(1 + 2 \cdot (1 + \Xi_0^{-1}) + 2)$$

$$= -5\Xi_0^0 + 13\Xi_0^{-1} - 40\Xi_0^{-2} + 120\Xi_0^{-3} + \ldots$$

(35)

Let us now extract the coefficient of $h = \Xi_0^{-1}$ from (35):

$$f_{,1}(x^0) = 13.$$

Second, let us calculate $f(x^0 + h e^2)$:

$$f(x^0 + h e^2) = f(1, 2 + \Xi_0^{-1})$$

$$= (1^2 + (2 + \Xi_0^{-1})^2)/(1 + 2 \cdot (2 + \Xi_0^{-1}))$$

$$= -5\Xi_0^0 + 10\Xi_0^{-1} - 20\Xi_0^{-2} + 20\Xi_0^{-3} + \ldots$$

(36)

Let us now extract the coefficient of $h = \Xi_0^{-1}$ from (36):

$$f_{,2}(x^0) = 1.$$

Finally, $\nabla f(x^0) = (f_{,1}(x^0), f_{,2}(x^0)) = (13, 1)$.

In Sect. 5, the proposed algorithm is applied for the com-
putation of the first variation $\delta e$ of the strain measure from
(14).

3.4 Computation of the Hessian

Let us first calculate the diagonal entries $H_{ii}(x^0)$, $i = 1, \ldots, m$, of the Hessian $H(x^0)$ from the second-order deri-
vatives of the function $y_i(t) = f(x^0 + t e^i)$ at $t = 0$ accord-
ing to (23). Again, the value $f(x^0 + h e^i)$ is calculated according
to (32). Then, the second-order partial derivative $f_{,ii}(x^0)$
(being the \(i\)-th diagonal entry \(H_{ij}(x^0)\) of the Hessian matrix \(H(x^0)\)) can be calculated extracting the coefficient \(a_2\) of \(h^2\):

\[
H_{ii}(x^0) = f_{,ii}(x^0) = 2a_2 \\
= 2 \text{FinitePart}[f(x^0 + he^i)h^{-2}],
\]

(37)

where \(e^i\) is the \(i\)-th column of the \(m\)-dimensional identity matrix, \(h\) is a numerical infinitesimal of the form (28), \(\text{FinitePart}[\cdot]\) is the finite part of the number \(x\), i.e., the coefficient of \(h^0\) or \(\Theta^0\) in (28). In other words, the \(i\)-th second-order partial derivative \(f_{,ii}(x^0)\) (being the \(i\)-th diagonal entry of \(H(x^0)\)) is the coefficient (multiplied by 2) of the infinitesimal \(h^2\) of the value \(f(x^0 + he^i)\) given in the form (28); if, e.g., \(h = \Theta^{-1}\), then the \(i\)-th diagonal entry of the Hessian can be calculated extracting the coefficient of \(\Theta^{-2}\) from \(f(x^0 + \Theta^{-1}e^i)\). An example of the computation of the derivatives in this way is presented below.

**Example 2** Let us calculate the second-order derivatives \(f_{,11}(x^0)\) and \(f_{,22}(x^0)\) of the function \(f(x) = \frac{x_1^2 + x_2^2}{x_1 + x_2 - 2x_1x_2}\) at \(x^0 = (1, 2)\). Let us fix the infinitesimal \(h\), e.g., equal to the simplest infinitesimal of the form (28): \(h = \Theta^{-1}\).

Let us first calculate \(f(x^0 + he^1)\):

\[
f(x^0 + he^1) = f(1 + \Theta^{-1}, 2) \\
= ((1 + \Theta^{-1})^2 + 2^2)/((1 + \Theta^{-1}) + 2) \\
= -5\Theta^{-1} + 13\Theta^{-2} - 40\Theta^{-3} + 120\Theta^{-4} + \ldots
\]

(38)

Let us now extract the coefficient of \(h^2 = \Theta^{-2}\) from (38):

\[
f_{,11}(x^0) = 2!(-40) = -80.
\]

Second, let us calculate \(f(x^0 + he^2)\):

\[
f(x^0 + he^2) = f(1, 2 + \Theta^{-1}) \\
= (1^2 + (2 + \Theta^{-1})^2)/(1 + (2 + \Theta^{-1})) \\
= -2\Theta^{-1} - 2\Theta^{-2} + 2\Theta^{-3} + \ldots
\]

(39)

Let us now extract the coefficient of \(h^2 = \Theta^{-2}\) from (39):

\[
f_{,22}(x^0) = 2!(-2) = -4.
\]

Let us now calculate the mixed second-order derivative \(f_{,ij}(x^0), \ i \neq j\), using the respective univariate derivatives of the function \(y_{ij}(t) = f(x^0 + te^i + te^j)\) at \(t = 0\) from (24). First, the value \(f(x^0 + he^i + he^j)\) is calculated:

\[
f(x^0 + he^i + he^j) = b_0 + b_1h + b_2h^2 + \ldots
\]

(40)

where \(b_k, \ k = 0, 1, \ldots\), are purely finite numbers. It should be noted that the coefficients \(b_k\) in (40) can be different from the coefficients \(a_k\) in (32), except \(a_0 = b_0 = f(x^0)\).

Second, the coefficient \(b_2\) is extracted from (40) and the second-order mixed derivative \(f_{,ij}(x^0)\) (being the \((i, j)\)-th entry \(H_{ij}(x^0)\) of the Hessian matrix \(H(x^0)\)) is calculated as follows:

\[
H_{ij}(x^0) = f_{,ij}(x^0) = b_2 - \frac{1}{2}[f_{,ii}(x^0) + f_{,jj}(x^0)] \\
= \text{FinitePart}[f(x^0 + he^i + he^j)h^{-2}] \\
= -\frac{1}{2}[H_{ii}(x^0) + H_{jj}(x^0)].
\]

(41)

Formally, the full algorithm for computation of the Hessian matrix is presented below.

**Algorithm 2** Computation of the Hessian matrix of the purely finite function \(f(x), \ f : \mathbb{R}^m \rightarrow \mathbb{R}\), at the purely finite point \(x^0 = (x_1^0, x_2^0, \ldots, x_m^0)\).

1: Fix the infinitesimal \(h\) in form (28).
2: for \(i = 1, 2, \ldots, m\) do
3: Calculate \(f(x^0 + he^i)\) according to (32),
4: Extract the coefficient \(a_2\) from (32),
5: Calculate \(H_{ii}(x^0)\) according to (37),
6: for \(j = 1, 2, \ldots, i - 1\) do
7: Calculate \(f(x^0 + he^i + he^j)\) according to (40),
8: Extract the coefficient \(b_2\) from (40),
9: Calculate \(H_{ij}(x^0) = H_{ji}(x^0)\) according to (41),
10: end for
11: end for
12: return \(H(x^0)\)

One can see that the Algorithm 2 is also easy to implement and parallelizable in different ways (e.g., along the \(i\)-based outer loop or along the \(j\)-based inner loop). Moreover, all partial derivatives \(f_{,jj}(x^0), \ j = 1, 2, \ldots, i\), used in the inner \(j\)-loop by (41) are known, since the inner \(j\)-loop calculates the \(i\)-th column of \(H(x^0)\) only up to the \(i\)-th row, i.e., using only known derivatives \(f_{,jj}(x^0)\). An example of the application of the algorithm is presented below.

**Example 3** Let us again consider the function \(f(x) = \frac{x_1^2 + x_2^2}{x_1 + x_2 - 2x_1x_2}\) and \(x^0 = (1, 2)\). Let us calculate the Hessian \(H(x^0)\) using \(h = \Theta^{-1}\).

The diagonal entries \(H_{ii}(x^0), \ i = 1, 2\) have been already calculated in the previous example: \(H_{11}(x^0) = -80\), \(H_{22}(x^0) = -4\).

In order to calculate the mixed derivative \(H_{12}(x^0) = f_{,12}(x^0)\), let us first calculate the value \(f(x^0 + he^1 + he^2) = f(1 + \Theta^{-1}, 2 + \Theta^{-1})\):

\[
f(x^0 + he^1 + he^2) = f(1 + \Theta^{-1}, 2 + \Theta^{-1})
\]
Remark 1 It should be noted that the Algorithms 1 and 2 require the computation of \( f \) within the same loop and using the same value \( f \).\dagger\ The gradient and the Hessian can be also calculated simultaneously if only the gradient is required, then \( f(x^0 + h e^i) \) can be calculated using only the powers of \( h \) up to 1 truncating all the powers greater than 1. If the Hessian is required, then \( f(x^0 + h e^i) \) (as well as \( f(x^0 + h e^j + h e^i) \)) should be calculated using the powers of \( h \) up to 2 truncating all the powers greater than 2.

Remark 2 One can also note that the gradient and the diagonal entries of the Hessian require different coefficients from the same value \( f(x^0 + h e^i) \) in (32). For this reason, the gradient and the Hessian can be also calculated simultaneously within the same loop and using the same value \( f(x^0 + h e^i) \) for both the \( i \)-th entry of the gradient and the \( i \)-th diagonal entry of the Hessian, if it is required to calculate them both. In this case, \( f(x^0 + h e^i) \) should be calculated using the powers of \( h \) up to 2 and no additional computation of \( f \) is required to complete the gradient.

Remark 3 Since \( H_{ij}(x^0) \) is calculated in (41) using subtractions, then the obtained values are calculated with a lower than the machine precision, if \( f(x) \) with its derivatives is ill-conditioned. For example, if the values \( H_{ii}(x^0) \) are too small, while \( H_{ij}(x^0) \) are large (or vice versa), then, due to ill-conditioning and limitations of the floating-point arithmetic, the results can lose significant digits. However, since the presented method does not use small finite stepsizes \( h \) (unlike standard finite difference schemes), then the obtained accuracy is still acceptable.

In Sect. 5, the proposed algorithm is applied for the computation of the second variation \( \delta \xi \) of the strain measure from (16).

4 Numerical results on benchmark problems

In this section, the proposed methodology for computation of the gradient and the Hessian is compared with several state-of-the-art methods on several widely used \( m \)-dimensional benchmark test problems from [40]: Shubert function (function \( f_1(x) \), \( m = 2 \)), Shekel function (function \( f_2(x) \), \( m = 4 \)) and Griewank functions (functions \( f_3(x) - f_5(x) \) with the dimensions \( m = 2, 4, 8, 16, 32, 64 \) and 128) over the interval \([1, 10]^m\) (the interval has been chosen in order to guarantee the twice continuous differentiability of the functions). The parameters of the functions and their implementation have been taken from the Virtual Library of Simulation Experiments http://www.sfu.ca/~ssurjano/.

- \( f_1(x) = \left( \sum_{i=1}^{5} \cos((i + 1)x_1 + i) \right) \),
- \( f_2(x) = \left( \sum_{i=1}^{10} \left( \sum_{j=1}^{4} (x_j - C_{ji})^2 + \beta_i \right) \right)^{-1} \),
- \( f_k(x) = \sum_{i=1}^{m} \frac{x_i^2}{4000} - \prod_{i=1}^{m} \cos \left( \frac{x_i}{\sqrt{i}} \right) + 1, \quad k = 3, 4, \ldots, 9. \) (45)

The following methods have been applied for differentiation of the functions given above:

- Adi—Automatic differentiation tool called “Adigator” from [55];
- TIC—The proposed method using the Infinity Computer for dealing with infinitesimals;
- Mat—The method based on the solution of a system of linear equations from [52] (the Infinity Computer was used for computation of the univariate derivatives up to machine precision);
- IS—Imaginary step method from [25].

As the automatic differentiation technique Adi is already exploited in the numerical experiments, the use of the software AceGen, discussed in the introduction, is not compared as it is based on automatic differentiation too and, thus, it will not improve the quality of results. Moreover, it requires the Wolfram Mathematica software adding so additional costs and difficulties for the authors of the present paper.

Numerical experiments have been carried out in MATLAB R2016b. For each test problem, symbolic derivatives have been calculated as reference values for the comparison of the algorithms. The infinitesimal perturbation for TIC has been set equal to the simplest infinitesimal \( \xi^{-1} \). The stepsize parameter \( h \) of the IS method has been set equal to \( 10^{-20} \) for the gradients according to [25], since this choice allows to calculate the gradient up to machine precision due to absence...
of cancellations. However, the computation of the Hessian using IS requires a more specific setting of $h$, since it involves subtractions and numerical cancellation for small $h$. For this reason, a parametric study of the impact of different values of the characteristic parameter $h$ (from $10^{-4}$ to $10^{-5}$) of the IS algorithm for computation of the Hessian is also carried out.

For each test function $f_k(x)$, $k = 1, \ldots, 9$, 50 trial points $x^i$, $i = 1, \ldots, 50$, have been generated randomly in the hyper-interval $[1, 10]^n$. The respective gradients $\nabla f_k(x^i)$ and Hessians $H_j f_k(x^i)$ have been calculated at $x^i$ by the algorithm $met$, $met \in \{Adi, TIC, Mat, IS\}$. The relative errors with respect to the symbolic derivatives $\nabla f_k(x^i)$ and $H(x^i)$ have been calculated componentwise as follows:

$$E_{grad}^{grad}(x^i) = \frac{|\nabla f_k(x^i) - \nabla f_k(x^i)|}{1 + |\nabla f_k(x^i)|}, \quad j = 1, \ldots, m.$$  

$$E_{hess}^{hess}(x^i) = \frac{|H_j f_k(x^i) - H_j f_k(x^i)|}{1 + |H_j f_k(x^i)|}, \quad j = 1, \ldots, m.$$  

The average relative errors have been calculated:

$$E_{avg}^{grad} = \frac{1}{50} \sum_{i=1}^{50} ||E_{grad}^{grad}(x^i)||_2, \text{ and } E_{avg}^{hess} = \frac{1}{50} \sum_{i=1}^{50} ||E_{hess}^{hess}(x^i)||_2.$$  

The obtained results are presented in Tables 1 and 2 for the gradients and Hessians, respectively.

One can see from Tables 1 and 2 that the proposed TIC algorithm allows to obtain the same accuracy as the automatic differentiation technique Adi: the obtained average relative error on the first test problem coincides for these two algorithms (being higher than the Machine precision due to the presence of small values in the derivatives), while the errors on the remaining functions are smaller than the machine precision for both the algorithms (the difference between $10^{-36}$ and $10^{-21}$ on the last function for the Hessian is non significant, since it is much smaller than the machine precision).

On the other hand, the algorithm Mat from [52] is able to obtain a high accuracy only on the problems with a relatively small dimension (functions 1–5 with the dimensions not higher than 8). It can be also seen that the accuracy of the method Mat decreases with the dimension of the function to be differentiated, while the accuracy of the proposed algorithm TIC does not depend on the dimensionality of the problem. Thus, the proposed algorithm TIC can be considered as an improvement of the “Mat” method showing so the desirable accuracy without concerns about the dimensionality of the problem.

Finally, the IS algorithm shows the same accuracy as Adi and TIC for the gradient, but lower accuracy for the Hessian. One can see that the accuracy of the IS method does not depend on the dimensionality of the problem as well as the TIC and Adi algorithms, but it involves numerical cancellation issues related to the finite step size $h$.

For the above mentioned reasons, the algorithms Adi and TIC produce equivalent accuracies. However, they are different from the computational point of view. The Adi algorithm overrides the arithmetic operations in the software code of the function $f(x)$ in order to construct the code for the respective derivatives. It is well-known that the generation of the code by Adi requires additional execution time, which increases with the complexity of the function to be differentiated (see, e.g., [21]). Consecutive evaluations of the derivatives depend on the complexity of the function $f(x)$ and its code. For complicated real-life examples where the function to be differentiated can change during the simulation, application of an automatic differentiation algorithm can require re-generation of the code for the derivatives every time when the function is changed. For this reason, its application requires not only temporal resources for the execution, but the resources for generation and storing of the respective derivatives.

In its turn, TIC algorithm does not involve generation of the code, but it uses the overridden operations similarly to the Adi algorithm. However, it does not require any additional space resources. For this reason, it does not re-generate the derivatives if the function changes. The latter property allows one to calculate the derivatives on-the-fly with the highest accuracy. A detailed comparison between TIC and Adi is not presented here, since it requires a fair implementation of both the methods. The Adi algorithm has been implemented in MATLAB for an efficient work with the matrices and vectors allowing vectorized operations, while TIC requires an efficient implementation of the framework for working with infinite and infinitesimal numbers. However, at this moment, only a preliminary non-optimized version of the Infinity Computer is available. This software simulator does

| # | Adi   | TIC   | Mat   | IS    |
|---|-------|-------|-------|-------|
| 1 | 2.0e−14 | 2.0e−14 | 4.2e−14 | 2.0e−14 |
| 2 | 1.8e−17 | 1.6e−17 | 4.8e−16 | 1.6e−17 |
| 3 | 1.2e−16 | 1.2e−16 | 2.0e−15 | 1.2e−16 |
| 4 | 1.1e−16 | 1.1e−16 | 1.4e−15 | 1.1e−16 |
| 5 | 2.9e−17 | 2.9e−17 | 5.5e−15 | 2.6e−17 |
| 6 | 1.6e−18 | 1.6e−18 | 2.3e−14 | 1.8e−18 |
| 7 | 1.4e−18 | 1.4e−18 | 1.4e−13 | 1.7e−18 |
| 8 | 5.2e−19 | 5.2e−19 | 2.5e−12 | 2.3e−18 |
| 9 | 8.6e−19 | 8.6e−19 | 5.0e−11 | 4.0e−18 |
Moreover, since the Infinity Computer with finite powers of 1 is analyzed and compared with its competitors. Studied where the computational performance of the Infinity Computer is proven to be “unstable” on the dimensions higher than 5.

Accurate similar to TIC, while the Mat algorithm has been involved in this section, since it produces results with the simulation. It should be noted that the Adi method is not adequate in the computation of the Hessian using IS requires a more accurate Hessian still enable the convergence but with a lower rate.

Instead, the IS method has been chosen for the comparison, since it uses only one additional framework being the complex numbers, which is well-implemented in different programming languages and environments (e.g., MATLAB or Python). Meanwhile, the accuracy of the IS algorithm for the gradient is high enough, while it can also be acceptable using “good” values of h for the Hessian. For these reasons, the IS algorithm can also be applicable for solving difficult real-life problems, so it is important to study the limits of its successful usage. Simulations using exact analytical formulae for the derivatives (AF) are also added as a reference solution for the comparison.

For each test problem, the parameter h of the IS method has been set equal to 10−20 for the gradients according to [25], since this choice allows to calculate the gradient up to machine precision due to absence of cancellations. However, the computation of the Hessian using IS requires a more specific setting of h, since it involves subtractions and numerical cancellation for small h (see Table 2). For this reason, a parametric study of the impact of different values of the characteristic parameter h of the IS algorithm for computation of the Hessian is also carried out. In this context, two numerical tolerances, 10−4 and 10−5, respectively, in solving the nonlinear equations are considered. The solution method is implemented in a predictor–corrector scheme based on a linear extrapolation of the previous solution step. Its length is modulated on the basis of the number of iterations required to reach the convergence. Then, if the step is correctly solved, the step-length grows along the less nonlinear part of the equilibrium path and decreases where the path is strongly nonlinear. When the discrete operators are affected by errors slowness in the nonlinear process can be observed.

### 5 Numerical results on computational mechanics

In this section, the efficiency of the proposed methodology for computation of the gradient and the Hessian of the nonlinear equations governing the static analysis of structures in large deformation problems is studied on several physical applications. Isogeometric Kirchhoff–Love shell model with C2 NURBS interpolation and patch-wise integration schemes from [31] is employed as a physical context of application. In this model, the accuracy of the simulation depends on the computation of the gradient involved in the evaluation of the structural response expressed by (10). The accuracy of the computation of the Hessian affects the convergence speed of the Newton algorithm. Small deviations with respect to the exact Hessian still enable the convergence but with a lower rate.

The proposed TIC methods are compared with the IS algorithm in terms of both the accuracy and the efficiency of the simulation. It should be noted that the Adi method is not involved in this section, since it produces results with the accuracy similar to TIC, while the Mat algorithm has been proven to be “unstable” on the dimensions higher than 5.

#### 5.1 Cantilever beam under compression

The first example is the simple Euler Beam test from [6,31]. The simple cantilever beam depicted in Fig. 1 is analyzed for different values of the slenderness parameters k = L/t,

### Table 2 Average relative errors in Hessians on 9 benchmark test functions

| #   | Adi      | TIC      | Mat      | IS, h |
|-----|----------|----------|----------|-------|
|     |          |          |          | 10−1  | 10−2  | 10−3  | 10−4  | 10−5  |
| 1   | 2.7e−14  | 2.7e−14  | 5.6e−12  | 1.9e+00| 1.8e−02| 1.8e−04| 1.8e−06| 4.6e−07|
| 2   | 2.6e−17  | 2.2e−17  | 1.5e−11  | 5.1e−04| 5.1e−06| 5.1e−08| 2.7e−08| 2.4e−06|
| 3   | 1.2e−16  | 1.2e−16  | 1.2e−13  | 9.4e−04| 9.4e−06| 9.4e−08| 2.2e−08| 2.0e−06|
| 4   | 1.4e−16  | 1.4e−16  | 6.2e−11  | 1.4e−03| 1.4e−05| 1.4e−07| 4.3e−08| 4.6e−06|
| 5   | 4.9e−17  | 5.1e−17  | 3.2e−10  | 6.6e−04| 6.6e−06| 6.6e−08| 9.7e−08| 9.2e−06|
| 6   | 1.9e−18  | 2.3e−18  | 6.9e−09  | 4.2e−05| 4.2e−07| 5.3e−09| 1.8e−07| 2.0e−05|
| 7   | 1.0e−19  | 1.3e−18  | 5.8e−08  | 1.9e−10| 3.9e−11| 1.8e−09| 2.8e−07| 5.0e−05|
| 8   | 2.6e−20  | 6.2e−19  | 2.6e−05  | 6.6e−13| 1.2e−10| 2.0e−09| 1.4e−06| 1.0e−04|
| 9   | 6.8e−36  | 2.4e−21  | 2.8e−02  | 1.9e−12| 4.6e−10| 4.0e−09| 3.2e−06| 2.1e−04|

![Image](image_url)
where $L$ and $t$ are the length and the thickness of the beam, respectively. A very small shear imperfection load ($\epsilon = 0.01$) is added to avoid the jump of the bifurcation as shown in [31].

The equilibrium path for different slenderness ratios, structural models, and values of $h$ of the IS method, are reported in Figs. 2, 3 and 4 for the $C^2$ NURBS. The load factor is normalized with respect to the analytical buckling load $\lambda_b$. The obtained iteration numbers for each model are presented in Table 3.

As it can be seen from Figs. 2, 3 and 4 and Table 3, the simulation using both AF and TIC gives always the lowest number of iterations keeping so the highest convergence speed of the Newton algorithm. However, since the IS calculates the Hessian using the finite stepsize $h$, its value influences significantly the performance of the IS algorithm in computation of the Hessian. In particular, one can see that the gradient is always calculated correctly, since the shape of the path obtained by IS in Figs. 2, 3 and 4 coincides with the shape obtained by TIC and AF. However, one can also see that different values of $h$ lead to different accuracies in Hessian for IS. In particular, for small values of $h$ (e.g., $h \leq 10^{-6}$), the simulation generates much more iterations, than the higher precision methods like TIC or AF, due to large errors in computation of the Hessian. In the case of $h = 10^{-6}$, the IS method is not usable and it is possible to observe that the behavior of the algorithm is optimal only at the beginning of the equilibrium path where the nonlinear aspects are not relevant yet. For this issue, the results obtained by IS with $h = 10^{-6}$ are presented in Figs. 2, 3 and 4 only for the case of tol $= 10^{-4}$ where it is also evident that for $L/t = 10$ the process is not able to track the whole equilibrium path.

It can be also noted that the IS method with bigger values of $h$ allows to obtain the same solution as AF or TIC for large values of $k = \frac{L}{t}$, while it generates more iterations for $k = 10$ also in this case. One can see that the results obtained by TIC coincide with the AF, since the proposed methods allow to calculate both the gradient and Hessian with a higher precision.

![Fig. 1 Euler beam: geometry and loads](image1)

![Fig. 2 Euler beam: equilibrium path for $L/h=10$ and $tol=1e-4$ (left) and $tol=1e-8$ (right). A part of the graph is zoomed in to improve the readability of the figures. The results coincide for AF (red points and dashed line) and TIC (red squares and red solid line) in both the cases; IS generates the same iterations only using $h = 1e-4$ (black triangles and lines) for $tol = 1e-4$ (left figure), while it generates different iterations w.r.t. AF using other values of $h$: IS with $h = 1e-2$ (blue triangles and lines) and $1e-3$ (green triangles and lines) and $1e-4$ with $tol = 1e-8$ don’t coincide with the analytical values obtained by AF; using $h = 1e-6$ (brown triangles and lines) it generates so many and too dense iterations even for $tol = 1e-4$, so it hasn’t been launched for $tol = 1e-8$. (Color figure online)](image2)
Fig. 3 Euler beam: equilibrium path for $L/h=100$ and $tol = 1e^{-4}$ (left) and $tol = 1e^{-8}$ (right). A part of the graph is zoomed in to improve the readability of the figures. The results coincide for AF (red points and dashed line) and TIC (red squares and red solid line) in both the cases; IS generates the same iterations only using $h = 1e^{-3}$ and $h = 1e^{-4}$ (green and black triangles and lines, respectively), while it generates different iterations w.r.t. AF using other $h = 1e^{-2}$ (blue triangles and lines) for $tol = 1e^{-8}$ and significantly more iterations using $h = 1e^{-6}$ (brown triangles and lines) even for $tol = 1e^{-4}$, so it hasn’t been launched for $tol = 1e^{-8}$. (Color figure online)

Fig. 4 Euler beam: equilibrium path for $L/h=1000$ and $tol = 1e^{-4}$ (left) and $tol = 1e^{-8}$ (right). A part of the graph is zoomed in to improve the readability of the figures. The results coincide for AF (red points and dashed line) and TIC (red squares and red solid line) in both the cases; IS generates the same iterations only using $h = 1e^{-3}$ and $h = 1e^{-4}$ (green and black triangles and lines, respectively), while it generates different iterations w.r.t. AF using other $h = 1e^{-2}$ (blue triangles and lines) for $tol = 1e^{-8}$ and significantly more iterations using $h = 1e^{-6}$ (brown triangles and lines) even for $tol = 1e^{-4}$, so it hasn’t been launched for $tol = 1e^{-8}$. (Color figure online)
Table 3  Euler beam: total number of iterations

| Method | tol $10^{-4}$ | tol $10^{-8}$ |
|--------|---------------|---------------|
|        | $L = 10$     | $L = 100$     | $L = 1000$ |
| AF     | 104           | 121           | 121        |
| TIC    | 104           | 121           | 121        |
| IS, $h$|               |               |            |
| $10^{-2}$ | 130           | 121           | 121        |
| $10^{-3}$ | 108           | 121           | 121        |
| $10^{-4}$ | 104           | 121           | 121        |
| $10^{-6}$ | 1217          | 196           | 187        |

Table 4  Composite shallow roof: material properties

| $E_{11}$ | $E_{22} = E_{33}$ | $v_{12} = v_{13} = v_{23}$ | $G_{12} = G_{13}$ | $G_{23}$ |
|----------|-----------------|----------------|----------------|---------|
| 3300     | 1100            | 0.25           | 660            | 440     |

5.2 Hinged cylindrical shallow roof

The benchmark depicted in Fig. 5 is analyzed in the multi-layered cases [0/90/0] and [90/0/90] with the material properties reported in Tab. 4 and thickness $t = 12.7$ (see [31] for details). The pointwise force is applied at the apex of the shell (point A). This test is particularly popular due to the snapping behaviour and because of that at some intermediate state, the tangential global stiffness matrices become singular with a subsequent unstable character. For this reason this test is a good candidate to check the accuracy in the evaluation of the second order derivatives.

In Figs. 6 and 7, the equilibrium paths are shown in terms of the load parameter $\lambda$ and $w_A$ for both the considered stacking sequences. The total number of iterations is also presented for each model in Table 5.

Also in this case, the results obtained by TIC coincide with the AF. However, one can also see that the IS gives acceptable results in this case without significant loss in efficiency, but still for small values of the parameter $h$ the method is neither accurate nor efficient. One can also see from Fig. 7(right) that the results obtained by IS with $h = 10^{-4}$ can differ from AF and TIC starting from some moment due to numerical errors in computation of the Hessian. Moreover, IS with $h = 10^{-6}$ generated too many iterations, so the analysis has not been completed due to the difficulties of the numerical solution process. Also in this case, the results of IS with $h = 10^{-6}$ are presented in Figs. 6 and 7 only for the case of tol $= 10^{-4}$ for the same reasons as previously stated.

Table 5  Composite shallow roof: total number of iterations

| Method | tol $10^{-4}$ | tol $10^{-8}$ |
|--------|---------------|---------------|
|        | [0/90/0]      | [90/0/90]     | [0/90/0]      | [90/0/90] |
| AF     | 97            | 64            | 298           | 141      |
| TIC    | 97            | 64            | 298           | 141      |
| IS, $h$| $10^{-2}$     | 97            | 64            | 298      | 141      |
| $10^{-3}$ | 97            | 64            | 298           | 141      |
| $10^{-4}$ | 97            | 64            | 298           | 141      |
| $10^{-6}$ | 4031          | 3868          | –             | –        |

Table 6  Clamped semi-cylinder: total number of iterations

| Method | tol $10^{-4}$ | tol $10^{-8}$ |
|--------|---------------|---------------|
| AF     | 138           | 762           |
| TIC    | 138           | 762           |
| IS, $h$| $10^{-2}$     | 138           | 762           |
| $10^{-3}$ | 138           | 762           |
| $10^{-4}$ | 164           | 926           |
| $10^{-6}$ | > 5000        | –             |

5.3 Clamped semi-cylinder

In Fig. 8, the geometry and the boundary conditions of another classical benchmark test from [53] are reported. The structure is a semi-cylinder loaded by a concentrated force at the middle of one of the curved edges (point A), while the other one is clamped. The vertical displacement of the straight edges is constrained. Due to its symmetry, only a half of the structure is analyzed. A composite multi-layered
Fig. 6 Composite shallow roof: equilibrium path for [0/90/0] $tol = 1e^{-4}$ (left) and $tol = 1e^{-8}$ (right). A part of the graph is zoomed in to improve the readability of the figures. The results coincide for AF (red points and dashed line) and TIC (red squares and red solid line) in both the cases; IS generates the same iterations only using the values of $h$ up to $1e^{-4}$, while it generates significantly more iterations using $h = 1e^{-6}$ (brown triangles and lines) even for $tol = 1e^{-4}$, so it hasn’t been launched for $tol = 1e^{-8}$. (Color figure online)

Fig. 7 Composite shallow roof: equilibrium path for [90/0/90] and $tol = 1e^{-4}$ (left) and $tol = 1e^{-8}$ (right). A part of the graph is zoomed in to improve the readability of the figures. The results coincide for AF (red points and dashed line) and TIC (red squares and red solid line) in both the cases; IS generates the same iterations only using the values of $h$ up to $1e^{-3}$, while it generates different iterations using $h = 1e^{-4}$ for $tol = 1e^{-8}$ and significantly more iterations using $h = 1e^{-6}$ (brown triangles and lines) even for $tol = 1e^{-4}$, so it hasn’t been launched for $tol = 1e^{-8}$. (Color figure online)
material is considered. The local reference system, used for defining the material properties, has the direction 1 aligned with the $y$ axis of the global system and the direction 3 is the normal to the surface from inside out. The stacking sequence of the laminated material is $[0/90/0]$, measured with respect to the direction 1 of the local reference system and the material properties are $E_1 = 2068.50$, $E_2 = E_3 = 517.125$, $v_{12} = v_{23} = v_{13} = 0.3$ and $G_{12} = G_{23} = G_{13} = 759.58$. This test is of interest in the shell context of analysis because the deformation is characterized by the development of wrinkles and a loss of accuracy could determine inaccuracy in the description of its nonlinear behaviour.

Also in this case the numerical experiments show that the results obtained by AF and TIC coincide, while the IS gives acceptable results only for large values of $h$. One can also see that inaccuracy in the evaluation of the second derivatives arises already for $h = 10^{-4}$. Moreover, one can see from Tables 3 and 6 that the best value of $h$ for IS depends on the problem and does not guarantee to generate the same solutions as TIC or AF do. For example, the IS using $h = 10^{-4}$ has generated less iterations (being so more efficient) than using other values of $h$ for the Euler Beam problem, while it performed significantly worse for the present example generating more iterations than AF or TIC. Again for smaller values of $h$ (e.g. $h = 10^{-6}$) the IS method is not applicable and with respect to the previous tests it seems particularly pathological. The solution process has been interrupted due to the high number of iterations nevertheless the equilibrium path have been tracked only at early part as it is shown in Fig. 9(left).

The equilibrium path in terms of the load parameter $\lambda$ and $w_A$ being the latter the displacement along $z$ of the point $A$ within the analysis is shown in Fig. 9.
6 Conclusion

A new numerical method for computation of the first and second order derivatives has been proposed within the framework for working numerically with infinite and infinitesimal numbers. The proposed method is simple and efficient allowing one to calculate the required derivatives with a high precision in both sequential and parallel ways. Moreover, it doesn’t require solution to ill-conditioned linear systems and doesn’t involve any small finite stepsize $h$ avoiding the issues related to numerical cancellation, being one of the main limitations for the numerical differentiation using finite differences. The proposed method allows to calculate the derivatives on-the-fly without neither pre-processing of the code nor analytical formulae for the functions. For these reasons, it becomes an effective numerical tool for real scale simulations.

It should be noted that the proposed approach is not limited by the Infinity Computer only and can be successfully used with alternative approaches for dealing with infinite and infinitesimal numbers (e.g., Levi-Civita field or hyper-dual numbers), so the computational effort of the proposed algorithms directly depends on the software implementation of the respective framework.

In particular, the application of this method in numerical analysis of structural behavior as black-box tool is useful due to its high accuracy and less error prone character that allow to avoid the analytical burden during the development of new mechanical models. Its use in other contexts of applied mathematics and physics is also desirable especially in coupled mechanical models. Its use in other contexts of applied mathematics and physics is also desirable especially in coupled mechanical models.

Appendix: Analytical strain variations of the full model

In this section, it is shown the shape of the first and second strain variations. Even though the mechanical model is yet quiet simple it is evident that the derivation process and the optimal shape for their use in a computational framework is not trivial. Since the strain variation of the membrane part is quadratic function of the displacements, only the strain variation of curvature is investigated in this paper. The curvature is expressed in the following convenient format, by using the spin operator $W$

$$\tilde{\chi}_{ij} = \left( \frac{g_{1j}g_{2}^T}{\|g_{1j}g_{2}\|} - G_{iaj}G_{3} \right) \quad \text{with} \quad i, j = 1, 2$$

By defining the following quantities

$$\delta \mathbf{b} = W[g_{1}]\delta \mathbf{d}_{2} - W[g_{2}]\delta \mathbf{d}_{1}$$

$$A_{ij} = \frac{a_{a_{2}}a_{3}}{\sqrt{\mathbf{b}^T\mathbf{b}}} = \frac{\mathbf{b}^T\mathbf{b} - a_{2}\mathbf{b}_{3}}{(\mathbf{b}^T\mathbf{b})},$$

the first variation of the curvature (being the gradient of the respective function) is

$$\delta \chi_{ij} = a_{3}^{T} \delta \mathbf{d}_{ij} + g_{ij}^{T} A_{ij} (W[g_{1}]\delta \mathbf{d}_{2} - W[g_{2}]\delta \mathbf{d}_{1}).$$

The second variation $\delta \tilde{\chi}_{ij}$ (being the Hessian of the respective function), involved in the construction of the iteration matrix of the Newton procedure can be calculated as follows:
\[ \dot{A}_{ij} = -\left(\dot{\mathbf{a}}_3 \dot{a}_3^T + a_3 \dot{\mathbf{a}}_3^T\right)(\mathbf{b}^T \mathbf{b})^{-1/2} - (I - a_3 \dot{\mathbf{a}}_3^T)(\mathbf{b}^T \mathbf{b})^{-3/2}(\mathbf{b}^T \mathbf{b}) \\
= -\frac{(\dot{a}_3 \dot{a}_3^T + a_3 \dot{\mathbf{a}}_3^T)}{\sqrt{\mathbf{b}^T \mathbf{b}}} - \frac{(I - a_3 \dot{\mathbf{a}}_3^T)}{(\mathbf{b}^T \mathbf{b}) \sqrt{\mathbf{b}^T \mathbf{b}}} \\
= -\frac{(\dot{a}_3 \dot{a}_3^T + a_3 \dot{\mathbf{a}}_3^T)}{\sqrt{\mathbf{b}^T \mathbf{b}}} - \frac{(b^T a_3)(I - a_3 a_3^T)}{b^T b}, \]

\[ \xi_{ij} = -\left(\frac{A_{ij} g_{ij}^T a_3^T + (g_{ij}^T a_3^T) A_{ij}}{\sqrt{b^T b}} + \frac{a_3 g_{ij}^T}{b^T b}(I - a_3 a_3^T)\right). \]

\[ \delta \dot{x}_{ij} = \dot{d}_{i}^T W_{g_2} A_{ij} g_{ij}^T \\
- \dot{d}_{j}^T W_{g_2} A_{ij} g_{ij}^T + g_{ij}^T A_{ij} W_{g_1} \delta d_2 \\
- \dot{g}_{ij} A_{ij} W_{g_2} \delta d_2 - \dot{d}_{j}^T W_{g_1} \xi_{ij} W_{g_1} \delta d_2 \\
- \dot{d}_{i}^T W_{g_2} \xi_{ij} W_{g_2} \delta d_2 \\
+ \dot{d}_{j}^T (W_{g_2} \xi_{ij} W_{g_1} - W_{A_{ij} g_{ij}^T}) \delta d_2 \\
+ \dot{d}_{i}^T (W_{g_1} \xi_{ij} W_{g_2} + W_{A_{ij} g_{ij}^T}) \delta d_2. \]

References

1. Amodio P, Brugnano L, Iavernaro F, Mazzia F (2020) On the use of the infinity computer architecture to set up a dynamic precision floating-point arithmetic. Soft Comput 24(23):17589–17600
2. Amodio P, Iavernaro F, Mazzia F, Mukhametzhanov MS, Sergeyev Y (2017) A generalized Taylor method of order three for the solution of initial value problems in standard and infinity floating-point arithmetic. Math Comput Simul 141:24–39
3. Aristodemo M (1985) A high-continuity finite element model for two-dimensional elastic problems. Comput Struct 21(5):987–993. https://doi.org/10.1016/0045-7949(85)90211-1
4. Baydin AG, Pearlmutter BA, Radul AA, Siskind M (2018) Automatic differentiation in machine learning: a survey. J Mach Learn Res 18:1–43
5. Bengi V, Di Nasso M (2003) Numerosities of labeled sets: a new way of counting. Adv Math 173:50–67
6. Caseiro J, Valente R, Reali A, Kiendl J, Auricchio F, Alves de Sousa R (2015) Assumed natural strain NURBS-based solid-shell formulation for geometrically non-linear analysis of large deformation elasto-plastic thin-shell structures. Comput Methods Appl Mech Eng 284:861–880. https://doi.org/10.1016/j.cma.2014.08.037
7. Cococcioni M, Cudazzo A, Pappalardo M, Sergeyev Y (2020) Solving the lexicographic multi-objective mixed-integer linear programming problem using branch-and-bound and grossone methodology. Commun Nonlinear Sci Numer Simul. https://doi.org/10.1016/j.cnsns.2020.105177
8. Cococcioni M, Pappalardo M, Sergeyev Y (2018) Lexicographic multiobjective linear programming using grossone methodology: theory and algorithm. Appl Math Comput 318:298–311
9. Cotrell JA, Hughes TJR, Bazilevs Y (2009) Isogeometric analysis: toward integration of CAD and FEA. 978-0-470-74873-2
10. De Cosmis S, De Leone R (2012) The use of grossone in mathematical programming and operations research. Appl Math Comput 218(16):8029–8038
11. De Leone R (2018) Nonlinear programming and grossone: quadratic programming and the role of constraint qualifications. Appl Math Comput 318:290–297
12. De Leone R, Fasano G, Sergeyev Y (2018) Planar methods and grossone for the conjugate gradient breakdown in nonlinear programming. Comput Optim Appl 71(1):73–93
13. Falcone A, Garro A, Mukhametzhanov MS, Sergeyev Y (2020) A Simulink-based software solution using the infinity computer methodology for higher order differentiation. Appl Math Comput. https://doi.org/10.1016/j.amc.2020.125606
14. Falcone A, Garro A, Mukhametzhanov MS, Sergeyev Y (2020) Representation of grossone-based arithmetic in simulink for scientific computing. Soft Comput 24(23):17525–17539. https://doi.org/10.1007/s00500-020-05221-y
15. Flynn DJ, Shamseddine K (2020) On computational applications of the Levi-Civita field. J Comput Appl Math 382:113041. https://doi.org/10.1016/j.cam.2020.113041
16. Garcea G, Trunfio G, Ciaciari R (1998) Mixed formulation and locking in path-following nonlinear analysis. Comput Methods Appl Mech Eng 165(1–4):247–272
17. Gaudioso M, Giallombardo G, Mukhametzhanov MS (2018) Numerical infinitesimals in a variable metric method for convex nonsmooth optimization. Appl Math Comput 318:312–320
18. Griewank A (1989) On automatic differentiation. In: Iri M, Tanabe K (eds) Mathematic programming: recent developments and applications. Kluwer Academic Publisher, pp 83–108
19. Herrerra AJ, Johnson EL, Proserpio D, Wu MCH, Kiendl J, Hsu MC (2018) Penalty coupling of non-matching isogeometric Kirchhoff–Love shell patches with application to composite wind turbine blades. Comput Methods Appl Mech Eng 346:810–840
20. Iavernaro F, Mazzia F, Mukhametzhanov MS, Sergeyev Y (2020) Conjugate-symplecticity properties of Euler–Maclaurin methods and their implementation on the infinity computer. Appl Numer Math 155:58–72
21. Iavernaro F, Mazzia F, Mukhametzhanov MS, Sergeyev Y (2021) Computation of higher order Lie derivatives on the infinity computer. J Comput Appl Math 383:113135
22. Kiendl J, Bletzinger KU, Linhard J, Wüchner R (2009) Isogeometric shell analysis with Kirchhoff–Love elements. Comput Methods Appl Mech Eng 198(49):3902–3914. https://doi.org/10.1016/j.cma.2009.08.013
23. Korelc J (2002) Multi-language and multi-environment generation of nonlinear finite element codes. Eng Comput 18(4):312–327
24. Korelc J, Wriggers P (2016) Automation of Finite Element Methods. Springer, Cham. https://doi.org/10.1007/978-3-319-39005-5
25. Lai KL, Crassidis J (2008) Extensions of the first and second complex-step derivative approximations. J Comput Appl Math 219(1):276–293
26. Lantoine G, Russell R, Dargenti T (2012) Using multicomplex variables for automatic computation of high-order derivatives. ACM Trans Math Softw 38(3):1–21
27. Leonetti L, Garcea G, Magisano D, Liguori F, Formica G, Lacarbonara W (2020) Optimal design of CNT-nanocomposite nonlinear shells. Nanomaterials. https://doi.org/10.3390/nano10122484
28. Leonetti L, Liguori F, Magisano D, Garcea G (2018) An efficient metric formulation of the Koiter’s theory for buckling and initial post-buckling analysis of elastic shells. Comput Methods Appl Mech Eng 331:159–183. https://doi.org/10.1016/j.cma.2017.11.025
29. Leonetti L, Liguori F, Magisano D, Kienzl J, Reali A, Garcea G (2020) A robust penalty coupling of non-matching isogeometric Kirchhoff–Love shell patches in large deformations. Comput Methods Appl Mech Eng 371:113289. https://doi.org/10.1016/j.cma.2020.113289
30. Leonetti L, Magisano D, Liguori F, Garcea G (2018) An isogeometric formulation of the Koiter’s theory for buckling and initial post-buckling analysis of composite shells. Comput Methods Appl Mech Eng 337:387–410. https://doi.org/10.1016/j.cma.2018.03.037

© Springer
31. Leonetti L, Magisano D, Madeo A, Garcea G, Kiendl J, Reali A (2019) A simplified Kirchhoff–Love large deformation model for elastic shells and its effective isogeometric formulation. Comput Methods Appl Mech Eng 354:369–396. https://doi.org/10.1016/j.cma.2019.05.025
32. Les Piegl WT (1997). The NURBS book. https://doi.org/10.1007/978-3-642-59223-2
33. Liguori FS, Zucco G, Madeo A, Garcea G, Leonetti L, Weaver PM (2019) An isogeometric framework for the optimal design of variable stiffness shells undergoing large deformations. Int J Solids Struct 210–211:18–34. https://doi.org/10.1016/j.ijsolstr.2020.11.003
34. Liguori FS, Zucco G, Madeo A, Magisano D, Leonetti L, Garcea G, Weaver PM (2019) Postbuckling optimisation of a variable angle tow composite wingbox using a multi-modal Koiter approach. Thin Walled Struct 138:183–198. https://doi.org/10.1016/j.tws.2019.01.035
35. Magisano D, Leonetti L, Garcea G (2017) Advantages of the mixed format in geometrically nonlinear analysis of beams and shells using solid finite elements. Int J Numer Methods Eng 109(9):1237–1262. https://doi.org/10.1002/nme.5322
36. Magisano D, Leonetti L, Garcea G (2021) Isogeometric analysis of 3d beams for arbitrarily large rotations: locking-free and path-independent solution without displacement DOFs inside the patch. Comput Methods Appl Mech Eng 373:113437. https://doi.org/10.1016/j.cma.2020.113437
37. Magisano D, Liguori F, Madeo A, Leonetti L, Garcea G (2021) Material design for optimal postbuckling behaviour of composite shells. Materials. https://doi.org/10.3390/ma14071665
38. Maurin F, Greco F, Dedoncker S, Desmet W (2018) Isogeometric analysis for nonlinear planar Kirchhoff rods: weighted residual formulation and collocation of the strong form. Comput Methods Appl Mech Eng 340:1023–1043. https://doi.org/10.1016/j.cma.2018.05.025
39. Maurin F, Greco F, Desmet W (2018) Isogeometric analysis for nonlinear planar pantographic lattice: discrete and continuum models. Contin Mech Thermodyn. https://doi.org/10.1007/s00161-018-0641-y
40. Momir J, Yang XS (2013) A literature survey of benchmark functions for global optimization problems. Int J Math Model Numer Optim 4(2):150–194
41. Mukhametzhanov MS, Sergeyev Y (2020) The infinity computer vs. symbolic computations: first steps in comparison. In: AIP conference proceedings, vol 2293, p 420045
42. Nguyen VP, Anitescu C, Borlas SPA, Rabczuk T (2015) Isogeometric analysis: an overview and computer implementation aspects. Math Comput Simul 117:89–116. https://doi.org/10.1016/j.matcom.2015.05.008
43. Riks E (1979) An incremental approach to the solution of snapping and buckling problems. Int J Solids Struct 15(7):529–551. https://doi.org/10.1016/0020-7683(79)90081-7
44. Robinson A (1996) Non-standard analysis. Princeton University Press, Princeton
45. Sergeyev Y (2010) Computer system for storing infinite, infinitesimal, and finite quantities and executing arithmetical operations with them. USA patent 7,860,914 (2010), EU patent 1728149 (2009), RF patent 2395111
46. Sergeyev Y (2003) Arithmetic of infinity. Edizioni Orizzonti Meridionali (2003, 2nd edn. 2013)
47. Sergeyev Y (2011) Higher order numerical differentiation on the infinity computer. Optim Lett 5(4):575–585
48. Sergeyev Y (2017) Numerical infinities and infinitesimals: methodology, applications, and repercussions on two Hilbert problems. EMS Surv Math Sci 4:219–320
49. Sergeyev Y, Kvasov D, Mukhametzhanov MS (2018) On strong homogeneity of a class of global optimization algorithms working with infinite and infinitesimal scales. Commun Nonlinear Sci Numer Simul 59:319–330
50. Sergeyev Y, Mukhametzhanov MS, Mazzia F, Iavernaro F, Amodio P (2016) Numerical methods for solving initial value problems on the infinity computer. Int J Unconv Comput 12(1):3–23
51. Sergeyev Y, Nasso M, Mukhametzhanov MS, Kvasov D (2021) Novel local tuning techniques for speeding up one-dimensional algorithms in expensive global optimization using Lipschitz derivatives. J Comput Appl Math 383:113134
52. Shamseddine K, Berz M (1996) Exception handling in derivative computation with non-archimedean calculus. In: Berz M, Bischof C, Corliss G, Griewank A Computational differentiation: techniques, applications, and tools. SIAM, Philadelphia, pp 37–51
53. Sze K, Liu X, Lo S (2004) Popular benchmark problems for geometric nonlinear analysis of shells. Finite Elem Anal Des 40(11):1551–1569. https://doi.org/10.1016/j.finel.2003.11.001
54. Tanaka M, Sasagawa T, Omote R, Fujikawa M, Balzani D, Schröder P (2016) Numerical methods for solving initial value problems on the infinity computer. Int J Unconv Comput 12(1):3–23
55. Weinstein M, Rao A (2017) Algorithm 984: Adigator, a toolbox for the algorithmic differentiation of mathematical functions in MATLAB using source transformation via operator overloading. ACM Trans Math Softw 44(2):1–25
56. Yu W, Blair M (2013) DNAD, a simple tool for automatic differentiation of Fortran codes using dual numbers. Comput Phys Commun 184(5):1446–1452

**Publisher’s Note** Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.