Applied Mathematics and Computation 445 (2023) 127856

Contents lists available at ScienceDirect

Applied Mathematics and Computation

journal homepage: www.elsevier.com/locate/amc

An FFT method for the numerical differentiation

Nadaniela Egidi*, Josephin Giacomini, Pierluigi Maponi, Michael Youssef

University of Camerino, School of Science and Technology, via Madonna delle Carceri 9, Camerino 62032 MC, Italy

ARTICLE INFO

Article history:
Received 4 August 2022
Revised 23 November 2022
Accepted 12 January 2023
Available online 25 January 2023

2022 MSC:
45C05
65D25
65R20

Keywords:
Differentiation
FFT
Integral Equation
Singular Value Expansion

ABSTRACT

We consider the numerical differentiation of a function tabulated at equidistant points. The proposed method is based on the Fast Fourier Transform (FFT) and the singular value expansion of a proper Volterra integral operator that reformulates the derivative operator. We provide the convergence analysis of the proposed method and the results of a numerical experiment conducted for comparing the proposed method performance with that of the Neville Algorithm implemented in the NAG library.

1. Introduction

We consider the problem of the numerical differentiation, where the derivative of a function has to be computed from the knowledge of the function values at equidistant points. This is a classical approximation problem with an important role in a wide range of applications such as: finance [1], image processing [2], inverse problems [3], parameter identification and numerical solution of ordinary and partial differential equations [4,5].

Unfortunately, the numerical differentiation is an ill-conditioned problem due to the unboundedness of the differentiation operator, so, even small errors, in the function values, can yield large errors in the computed numerical derivative. However, due to its importance in the applied sciences, several methods have been proposed for the stable computation of first (and higher) order derivatives [6,7]; for instance we mention: finite-difference approaches [8,9]; kernel-based numerical differentiation methods [10]; interpolation methods [11]; random samplings for numerical differentiation in many variables [12]. A more detailed discussion on the stability issues of the numerical differentiation problem and possible regularization strategies can be found in [9,13–16].

In [9,17,18], the numerical differentiation problem has been reformulated as a Volterra integral equation. More precisely, in [18] the numerical differentiation of functions specified by data affected by noise is regularized by the truncated singular value decomposition. In [17] the special case of periodic functions specified by finite noisy data is considered and a truncated Fourier series technique is used to filter high frequency components of the spectral derivatives. Moreover, in [19] the

* Corresponding author.
E-mail addresses: nadaniela.egidi@unicam.it (N. Egidi), josephin.giacomini@unicam.it (J. Giacomini), pierluigi.maponi@unicam.it (P. Maponi), michael.youssef@studenti.unicam.it (M. Youssef).

https://doi.org/10.1016/j.amc.2023.127856
0096-3003/

derivatives of functions on an arbitrary interval are obtained by extending these functions to the whole real axis and regularizing these extensions. In [20] the numerical differentiation problem is transformed into a Fredholm integral equation of the first kind and solved by the Galerkin method with a trigonometric basis. In [21], the Legendre expansion is used for computing numerical derivatives of a function from its noisy data and ill posedness is treated by combining Tikhonov regularization with a new penalty term. In [22], the numerical differentiation problem with noisy data is regularized by using the Volterra integral equation.

In this paper we propose a new method for the numerical differentiation. This method is based on the singular value expansion (SVE) of the Volterra integral operator similar to the one considered in [9,17,18,22], and the Fast Fourier Transform (FFT) to approximate such an expansion. The resulting algorithm has been analysed in terms of the convergence rate and tested on some functions, by comparing the numerical results with those obtained by an extension of the Neville Algorithm implemented in the NAG library [23].

In Section 2 we describe the Volterra integral equation associated with the derivative problem and the corresponding SVE. In Section 3 the SVE of the Volterra integral operator and the FFT are used to define the method for the numerical differentiation method, and an error estimation for this method is provided. In Section 4 we present two algorithms based on the method introduced in Section 3: *FOD* and *NOD* that approximate the first order and ν-order derivative, respectively, of a given function in equidistant points. In Section 5 we describe the results of a numerical experiment with the proposed algorithms. In Section 6 some observations and future developments are discussed.

2. The SVE for the differentiation operator

We reformulate the differentiation operator as a suitable Volterra integral equation of first kind and we describe its SVE.

2.1. The general case

Let $\nu \geq 1$ and $f^{(j)}$, $j = 0, \ldots, \nu$, be the jth continuous derivative of $f : [0, 1] \to \mathbb{R}$. Suppose that we already know or have already calculated $f^{(j)}(0)$, $j = 0, \ldots, \nu - 1$. In [24], it has been proved that $v = f^{(\nu)}$ is the unique solution of the following integral equation

$$\int_0^1 K_\nu(x, y)v(y)dy = f(x) - \sum_{j=0}^{\nu-1} \frac{f^{(j)}(0)}{j!}x^j, \quad x \in [0, 1], \tag{1}$$

where

$$K_\nu(x, y) = \begin{cases} \frac{(x-y)^{\nu-1}}{(\nu-1)!}, & 0 \leq y < x \leq 1, \\ 0, & 0 \leq x \leq y \leq 1. \end{cases} \tag{2}$$

Let $\mathcal{K}_\nu$ be the integral operator associated to (2), we call $\mathcal{K}_\nu$ the ν-order operator and Eq. (1) can be rewritten as

$$\mathcal{K}_\nu v = g_\nu, \tag{3}$$

where g_ν is the known function

$$g_\nu(x) = f(x) - \sum_{j=0}^{\nu-1} \frac{f^{(j)}(0)}{j!}x^j, \quad x \in [0, 1]. \tag{4}$$

The knowledge of the second addendum in (4) is a natural assumption, in fact, when we compute the first derivative we suppose to know f, moreover, when $\nu > 1$ the procedure for the computation of the ν-derivative can be previously used to compute $f^{(j)}(0)$, $j = 0, \ldots, \nu - 1$. So ultimately only the function values are required in the computation of $f^{(\nu)}$.

In [25] the SVE of the kernel (2) has been computed and we briefly show these results below. Let $\mu_0 \geq \mu_1 \geq \ldots > 0$, be the singular values of K_ν. We denote with u_l and v_l, $l = 0, 1, \ldots$, the left-singular function and right-singular function, respectively, associated with μ_l, then the SVE of $\mathcal{K}_\nu$ is

$$K_\nu(x, y) = \sum_{l=0}^{\infty} \mu_l u_l(x)v_l(y), \quad x, y \in [0, 1]. \tag{5}$$

For $k \in \mathbb{N}$, let $\rho_2(k)$ be the reminder of the division of k by 2.

For $q \in \mathbb{Z}$ and $\gamma \in \mathbb{R}$ we define:

$$\begin{aligned} \theta_q &= \begin{cases} \frac{q\pi}{\nu}, & \text{if } \nu \text{ is even}, \\ \frac{(2q+1)\pi}{2\nu}, & \text{if } \nu \text{ is odd}, \end{cases} \\ s_q &= \sin(\theta_q), \qquad c_q = \cos(\theta_q), \\ s_{k,q}^{(\gamma)} &= \sin((k+1)\theta_q - \gamma s_q), \\ c_{k,q}^{(\gamma)} &= \cos((k+1)\theta_q - \gamma s_q), \end{aligned}$$

$$\alpha_q^{(\gamma)} = (-1)^q e^{\gamma c_q}.$$

The computation of the singular values and the singular functions of the ν-order operator can be obtained from the following two theorems.

Theorem 2.1. *Let $\gamma_l = 1/\sqrt[\nu]{\mu_l}$, where $\mu_l > 0$ is a singular value of $\mathcal{K}_\nu$. The singular functions corresponding to μ_l are*

$$u_l(x) = \sum_{p=0}^{\nu-\rho_2(\nu)} e^{\gamma_l c_p x}\left(C_p^{(u)}\cos(\gamma_l s_p x) + S_p^{(u)}\sin(\gamma_l s_p x)\right),\ x \in [0,1], \tag{6}$$

$$v_l(x) = \sum_{p=0}^{\nu-\rho_2(\nu)} e^{\gamma_l c_p x}\left(C_p^{(v)}\cos(\gamma_l s_p x) + S_p^{(v)}\sin(\gamma_l s_p x)\right),\ x \in [0,1], \tag{7}$$

where, for $p = 0, 1, \dots, \nu - \rho_2(\nu)$, the coefficients $S_p^{(\cdot)}, C_p^{(\cdot)} \in \mathbb{R}$, are solutions of the followings:

- *if ν is odd*

$$S_p^{(u)} = (-1)^p C_p^{(v)},\ C_p^{(u)} = (-1)^{p+1} S_p^{(v)}, \tag{8}$$

$$\sum_{p=0}^{\nu-1} \alpha_p^{(\gamma_l)}\left(S_p^{(v)} c_{k,p}^{(\gamma_l)} + C_p^{(v)} s_{k,p}^{(\gamma_l)}\right) = 0,\ k = 0, 1, \dots, \nu - 1, \tag{9}$$

$$\sum_{p=0}^{\nu-1}\left(C_p^{(v)} c_{k,p}^{(0)} - S_p^{(v)} s_{k,p}^{(0)}\right) = 0,\ k = 0, 1, \dots, \nu - 1; \tag{10}$$

- *if ν is even*

$$S_0^{(u)} = S_\nu^{(u)} = S_0^{(v)} = S_\nu^{(v)} = 0, \tag{11}$$

$$S_p^{(u)} = (-1)^p S_p^{(v)},\ C_p^{(u)} = (-1)^p C_p^{(v)}, \tag{12}$$

$$\sum_{p=0}^{\nu} \alpha_p^{(\gamma_l)}\left(C_p^{(v)} c_{k,p}^{(\gamma_l)} - S_p^{(v)} s_{k,p}^{(\gamma_l)}\right) = 0,\ k = 0, 1, \dots, \nu - 1, \tag{13}$$

$$\sum_{p=0}^{\nu}\left(C_p^{(v)} c_{k,p}^{(0)} - S_p^{(v)} s_{k,p}^{(0)}\right) = 0,\ k = 0, 1, \dots, \nu - 1. \tag{14}$$

Proof. See [25].□

Theorem 2.2. *Let $M(\gamma) \in \mathbb{R}^{2\nu\times 2\nu}$ be the coefficients matrix of linear system (9)-(10) when ν is odd or of linear system (13)-(14) when ν is even. Let μ_l, $l = 0, 1, \dots,$ be the singular values of $\mathcal{K}_\nu$, then $\gamma = \gamma_l = 1/\sqrt[\nu]{\mu_l}$ are the positive zeros of*

$$h_\nu(\gamma) = \det(M(\gamma)), \qquad \gamma \in \mathbb{R}.$$

When $\nu = 1$ we have

$$h_1(\gamma) = -\cos(\gamma). \tag{15}$$

Proof. See [25].□

The Theorems 2.1 and 2.2 provide the formulas for the computation of the SVE of the ν-order operator. Therefore, from standard arguments of mathematical analysis, the solution of (3) is given by

$$f^{(\nu)}(x) = \sum_{l=0}^{\infty} \frac{1}{\mu_l}\langle g_\nu, u_l\rangle v_l(x), \qquad x \in (0,1), \tag{16}$$

where $\langle g_\nu, u_l\rangle = \int_0^1 g_\nu(x) u_l(x) dx$.

2.2. The case of first order derivative

We restrict our attention to the case $\nu = 1$ and the material discussed in the previous section is analysed for the first derivative.

The first derivative f' of f is the unique solution $v = f'$ of the integral equation

$$\int_0^1 K_1(x,y)v(y)dy = f(x) - f(0), \quad x \in [0,1], \tag{17}$$

and $K_1 : [0,1] \times [0,1] \to \mathbb{R}$ is

$$K_1(x,y) = \begin{cases} 0, & 0 \le x \le y \le 1, \\ 1, & 0 \le y < x \le 1. \end{cases} \tag{18}$$

From Theorem 2.2, when $\nu = 1$, the singular values μ_l, $l = 0, 1, \dots$, are $\mu_l = \frac{1}{\gamma_l}$ where γ_l is a zero of the function

$$h_1(\gamma) = -\cos(\gamma). \tag{19}$$

So

$$\gamma_l = \left(l + \frac{1}{2}\right)\pi, \qquad l = 0, 1, \dots, \tag{20}$$

$\mu_l = \frac{1}{\gamma_l}$, and the singular functions $u_l(x)$, $v_l(x)$, $l = 0, 1, \dots$, associated with the singular values μ_l are computed by using Theorem 2.1, that gives

$$u_l(x) = \sqrt{2}\sin(\gamma_l x), \quad v_l(x) = \sqrt{2}\cos(\gamma_l x). \tag{21}$$

From (16), the derivative of f, that is the solution of (17), is given by

$$f'(x) = \sum_{l=0}^{\infty} \gamma_l \langle g, u_l \rangle v_l(x), \qquad 0 < x < 1, \tag{22}$$

where

$$g(x) = g_1(x) = f(x) - f(0). \tag{23}$$

2.3. The case of higher order derivatives

The integral operator associated to the first order derivative can be used to factorize the integral operators of higher order derivatives. Here we report the relation between $\mathcal{K}_2$ and $\mathcal{K}_1$,

$$\begin{aligned}(\mathcal{K}_2\phi)(x) &= \int_0^1 K_2(x,y)\phi(y)dy = \int_0^x (x-y)\phi(y)dy = \int_0^x \left(\int_y^x \phi(y)dt\right)dy = \int_0^x \left(\int_0^t \phi(y)dy\right)dt = \\ &= \int_0^x (\mathcal{K}_1\phi)(t)dt = (\mathcal{K}_1\mathcal{K}_1\phi)(x) = \left(\mathcal{K}_1^2\phi\right)(x),\end{aligned}$$

and generally for $\nu \ge 2$

$$\mathcal{K}_\nu\phi = \mathcal{K}_1^\nu\phi.$$

Moreover from (17) for $\nu \ge 1$ we have

$$\mathcal{K}_1 f^{(\nu)} = f^{(\nu-1)} - f^{(\nu-1)}(0),$$

and from $f^{(\nu-1)}$ we can compute $f^{(\nu)}$ by using formula (22) with $g(x) = f^{(\nu-1)}(x) - f^{(\nu-1)}(0)$. Note that in the last formula we assume $f^{(0)} = f$.

3. An FFT approach for the SVE computation in the case $\nu = 1$

For $l, k, j \in \mathbb{Z}$, and $h \in \mathbb{R}$, $h > 0$, we define

$$x_k = \left(k + \frac{1}{2}\right)h, \tag{24}$$

$$\xi_j = jh, \tag{25}$$

$$c_{l,k} = \cos\left(\left(l + \frac{1}{2}\right)\left(k + \frac{1}{2}\right)\pi h\right), \tag{26}$$

$$s_{l,k} = \sin\left(\left(l + \frac{1}{2}\right)\left(k + \frac{1}{2}\right)\pi h\right), \tag{27}$$

$$\tilde{s}_{l,k} = \sin\left(\left(l + \frac{1}{2}\right)k\pi h\right). \tag{28}$$

We note that the quantities given above depend on the choice of h, but in the remainder of this section we suppose that $h = 1/n$, for a given integer number $n > 1$, and for $l = 0, \ldots, n-1$, and $k \in \mathbb{Z}$, we have

$$0 = \xi_0 < x_0 < \xi_1 < x_1 < \xi_2 < \ldots < \xi_{n-1} < x_{n-1} < \xi_n = 1, \tag{29}$$

$$\sqrt{2}c_{l,k} = v_l(x_k) = v_k(x_l), \qquad \sqrt{2}s_{l,k} = u_l(x_k) = u_k(x_l), \tag{30}$$

$$\sqrt{2}\tilde{s}_{l,k} = u_l(\xi_k), \qquad \tilde{s}_{l,0} = 0, \qquad \tilde{s}_{l,n} = (-1)^l. \tag{31}$$

For $l = 0, 1, \ldots,$ we use the following notations

$$g_{l,u} = \langle g, u_l \rangle, \qquad g_{l,v} = \langle g, v_l \rangle, \tag{32}$$

$$g_{l,v}^{(k)} = \langle g^{(k)}, v_l \rangle, \qquad g_{l,u}^{(k)} = \langle g^{(k)}, u_l \rangle, \qquad k \geq 0, \tag{33}$$

where for $k = 0$ we assume $g^{(0)} = g$ so that $g_{l,v}^{(0)} = g_{l,v}$ and $g_{l,u}^{(0)} = g_{l,u}$.

Proposition 3.1. *For $l = 0, 1, \ldots,$ and $k > 0$ the following relations hold:*

$$\gamma_l g_{l,v}^{(k-1)} = g^{(k-1)}(1)u_l(1) - g_{l,u}^{(k)}, \tag{34}$$

$$\gamma_l g_{l,u}^{(k-1)} = g^{(k-1)}(0)v_l(0) + g_{l,v}^{(k)}. \tag{35}$$

Proof. They are simple consequences of the integration by parts formula.□

We note that from (35) and using the fact that $g(0) = 0$ (see formula (23)) we have

$$\gamma_l g_{l,u} = g_{l,v}^{(1)}, \qquad l = 0, 1, \ldots. \tag{36}$$

Let $\underline{w} = (w_0, w_1, \ldots, w_{n-1})^t \in \mathbb{R}^n$ and

$$\hat{w}_k = \sqrt{\frac{2}{n}}\sum_{l=0}^{n-1} w_l c_{l,k}, \qquad k = 0, 1, \ldots, n-1, \tag{37}$$

then $\underline{\hat{w}} = (\hat{w}_0, \hat{w}_1, \ldots, \hat{w}_{n-1})^t \in \mathbb{R}^n$ is the discrete cosine transform of type 4 of $\underline{w}$ and we write

$$\underline{\hat{w}} = DCT^{(4)}(\underline{w}).$$

Let $\underline{z} = (z_1, z_2, \ldots, z_n)^t \in \mathbb{R}^n$ and

$$\tilde{z}_k = \sqrt{\frac{1}{n}}\left(2\sum_{l=1}^{n-1} z_l \tilde{s}_{k,l} + (-1)^k z_n\right), \qquad k = 0, 1, \ldots, n-1, \tag{38}$$

then $\underline{\tilde{z}} = (\tilde{z}_0, \tilde{z}_1, \ldots, \tilde{z}_{n-1})^t \in \mathbb{R}^n$ is the discrete sine transform of type 3 of $\underline{z}$ and we write

$$\underline{\tilde{z}} = DST^{(3)}(\underline{z}).$$

We note that $DCT^{(4)}$ is an involutory operator, that is $DCT^{(4)}\big(DCT^{(4)}(\underline{w})\big) = \underline{w}$. Moreover, from (22) and (30), for $k = 0, 1, \ldots, n-1$, we have

$$g'(x_k) = \sqrt{2}\sum_{l=0}^{\infty} \gamma_l g_{l,u} c_{l,k}, \tag{39}$$

and, by truncating the above series, we have the following approximation of g' (and hence of f') at x_k, $k = 0, 1, \ldots, n-1$,

$$(g'(x_0), g'(x_1), \ldots, g'(x_{n-1}))^t \approx DCT^{(4)}(\sqrt{n}\underline{g}'_v), \tag{40}$$

where

$$\underline{g}'_v = (\gamma_0 g_{0,u}, \gamma_1 g_{1,u}, \ldots, \gamma_{n-1} g_{n-1,u})^t \in \mathbb{R}^n. \tag{41}$$

In order to explain the above used notation $\underline{g}'_\nu$, we note that, from (36), we have

$$\underline{g}'_\nu = (g^{(1)}_{0,\nu}, g^{(1)}_{1,\nu}, \ldots, g^{(1)}_{n-1,\nu})^t = (\langle g', \nu_0\rangle, \langle g', \nu_1\rangle, \ldots, \langle g', \nu_{n-1}\rangle)^t.$$

The proposed method consists in the use of the discrete sine transform of type 3 to compute an approximation $\underline{g}^p_\nu$ of $\underline{g}'_\nu$ and then compute an approximation $\underline{g}^p$ of $(g'(x_0), g'(x_1), \ldots, g'(x_{n-1}))^t$ from (40). More precisely,

$$\underline{g}^p = (g^p_0, g^p_1, \ldots, g^p_{n-1})^t = DCT^{(4)}(\sqrt{n}\underline{g}^p_\nu), \tag{42}$$

where $\underline{g}^p_\nu = (g^p_{0,\nu}, g^p_{1,\nu}, \ldots, g^p_{n-1,\nu})^t \in \mathbb{R}^n$ and for $l = 0, 1, \ldots, n-1$,

$$\begin{aligned} g^p_{l,\nu} = \frac{\sqrt{2}}{24}\Big[&\sqrt{n}(DST^{(3)}(\underline{g}))_l(27s_{l,0} - s_{l,1}) + c_{l,0}(-5g_1 + 4g_2 - g_3) + \\ &+ c_{l,n-1}(g_{n-3} - 4g_{n-2} + 7g_{n-1} - 4g_n)\Big] \end{aligned} \tag{43}$$

where $\underline{g} = (g_1, \ldots, g_n)^t = (g(\xi_1), \ldots, g(\xi_n))^t \in \mathbb{R}^n$ are the data at equispaced points. We remind that $g_0 = g(0) = 0$ and $g_n = g(\xi_n) = g(1)$.

The following theorem shows a convergence result for the quantities defined in formulas (42) and (43).

Theorem 3.2. *For $k = 0, 1, \ldots, n-1$, g^p_k, given in (42), is an approximation of $g'(x_k)$, in particular for a sufficiently regular function g we have*

$$g'(x_k) - g^p_k = \mathcal{O}(h^4), \qquad h \to 0. \tag{44}$$

Proof. From (43), by adding and subtracting appropriate quantities and observing that $c_{l,n-1} = -c_{l,n}$ and $g(\xi_0) = 0$, we have

$$\begin{aligned} g^p_{l,\nu} = \frac{\sqrt{2}}{24}\Big[&\sqrt{n}(DST^{(3)}(\underline{g}))_l(27s_{l,0} - s_{l,1}) + \\ &+\Big(-23g(\xi_0) + 21g(\xi_1) + 3g(\xi_2) - g(\xi_3) - 26g(\xi_1) + g(\xi_2)\Big)c_{l,0} - \\ &+\Big(g(\xi_{n-3}) - 3g(\xi_{n-2}) - 21g(\xi_{n-1}) + 23g(\xi_n) - g(\xi_{n-2}) + \\ &+27g(\xi_{n-1}) - 27g(\xi_n)\Big)c_{l,n-1} - g(\xi_{n-1})c_{l,n}\Big]. \end{aligned} \tag{45}$$

From Taylor expansion, we have

$$-23g(\xi_0) + 21g(\xi_1) + 3g(\xi_2) - g(\xi_3) = 24h\tilde{g}^p_0, \tag{46}$$

$$g(\xi_{n-3}) - 3g(\xi_{n-2}) - 21g(\xi_{n-1}) + 23g(\xi_n) = 24h\tilde{g}^p_{n-1}, \tag{47}$$

where

$$\tilde{g}^p_0 = g'(x_0) + \mathcal{O}\big(h^4\big), \qquad \tilde{g}^p_{n-1} = g'(x_{n-1}) + \mathcal{O}\big(h^4\big), \qquad h \to 0, \tag{48}$$

in particular $\tilde{g}^p_0$ and $\tilde{g}^p_{n-1}$ are approximations of g' at x_0 and x_{n-1}, respectively. By substituting (46) and (47) into expression (45) and using (38), we obtain

$$\begin{aligned} g^p_{l,\nu} = \frac{\sqrt{2}}{n}\Big[&\tilde{g}^p_0 c_{l,0} + \tilde{g}^p_{n-1}c_{l,n-1} + \\ &+\frac{n}{24}\Big[-26g(\xi_1)c_{l,0} + g(\xi_2)c_{l,0} + \sqrt{n}(DST^{(3)}(\underline{g}))_l\big(27s_{l,0} - s_{l,1}\big) + \\ &\quad -g(\xi_{n-2})c_{l,n-1} + g(\xi_{n-1})(27c_{l,n-1} - c_{l,n}) - 27g(\xi_n)c_{l,n-1}\Big]\Big] = \\ = \frac{\sqrt{2}}{n}\Big[&\tilde{g}^p_0 c_{l,0} + \tilde{g}^p_{n-1}c_{l,n-1} + \\ &+\frac{n}{24}\Big[-2\sum_{k=1}^{n-1} g(\xi_k)\tilde{s}_{l,k}s_{l,1} - g(\xi_n)\tilde{s}_{l,n}s_{l,1} + \\ &\quad +g(\xi_1)c_{l,0} + g(\xi_2)c_{l,0} - g(\xi_{n-2})c_{l,n-1} - g(\xi_{n-1})c_{l,n}\Big] + \\ &+\frac{27n}{24}\Big[2\sum_{k=1}^{n-1} g(\xi_k)\tilde{s}_{l,k}s_{l,0} + g(\xi_n)\tilde{s}_{l,n}s_{l,0} + \end{aligned}$$

$$-g(\xi_1)c_{l,0}+g(\xi_{n-1})c_{l,n-1}-g(\xi_n)c_{l,n-1}\Big]\Big].$$

By substituting in the above identity the following relations

$$\begin{aligned}
&c_{l,k+1}-c_{l,k-2}=-2\tilde{s}_{l,k}s_{l,1},\\
&c_{l,k-1}-c_{l,k}=2\tilde{s}_{l,k}s_{l,0},\\
&c_{l,-1}=c_{l,0},\\
&c_{l,n-2}=\tilde{s}_{l,n}s_{l,1},\\
&c_{l,n-1}=\tilde{s}_{l,n}s_{l,0},\\
&\tilde{s}_{l,n}=(-1)^l,\qquad \tilde{s}_{l,0}=0,
\end{aligned}$$

we obtain

$$\begin{aligned}
g^p_{l,\nu}=\frac{\sqrt{2}}{n}\Bigg[&\tilde{g}^p_0c_{l,0}+\tilde{g}^p_{n-1}c_{l,n-1}+\\
&+\frac{n}{24}\Bigg[\sum_{k=1}^{n-1}g(\xi_k)\big(c_{l,k+1}-c_{l,k-2}\big)-g(\xi_n)c_{l,n-2}+\\
&\qquad+g(\xi_1)c_{l,0}+g(\xi_2)c_{l,0}-g(\xi_{n-2})c_{l,n-1}-g(\xi_{n-1})c_{l,n}\Bigg]+\\
&+\frac{27n}{24}\Bigg[\sum_{k=1}^{n-1}g(\xi_k)\big(c_{l,k-1}-c_{l,k}\big)+g(\xi_n)c_{l,n-1}+\\
&\qquad-g(\xi_1)c_{l,0}+g(\xi_{n-1})c_{l,n-1}-g(\xi_n)c_{l,n-1}\Bigg]\Bigg]=\\
=\frac{\sqrt{2}}{n}\Bigg[&\tilde{g}^p_0c_{l,0}+\tilde{g}^p_{n-1}c_{l,n-1}+\\
&+\frac{n}{24}\Bigg[\sum_{k=1}^{n-2}\Big(g(\xi_{k-1})-27g(\xi_k)+27g(\xi_{k+1})-g(\xi_{k+2})\Big)c_{l,k}\Bigg]\Bigg]=\\
=\frac{1}{\sqrt{n}}&\Big(DCT^{(4)}\big(\underline{\tilde{g}}^p\big)\Big)_l,
\end{aligned}\tag{49}$$

where for $k=1,2,\dots,n-2$, we have defined $\tilde{g}^p_k$ such that

$$24h\tilde{g}^p_k=g(\xi_{k-1})-27g(\xi_k)+27g(\xi_{k+1})-g(\xi_{k+2}),\tag{50}$$

and from the Taylor expansion of g we can prove that

$$\tilde{g}^p_k=g'(x_k)+\mathcal{O}(h^4)\qquad h\to 0.\tag{51}$$

Given the vector

$$\underline{\tilde{g}}^p=(\tilde{g}^p_0,\tilde{g}^p_1,\dots,\tilde{g}^p_{n-1})\in\mathbb{R}^n,$$

whose components are defined in (46), (47) and (50), identity (49) becomes

$$\underline{g}^p_\nu=\frac{1}{\sqrt{n}}DCT^{(4)}\big(\underline{\tilde{g}}^p\big).\tag{52}$$

Finally, from (42), (48), (51) and (52) and the involutory property of $DCT^{(4)}$ we have for $k=0,1,\dots,n-1$,

$$\begin{aligned}
g'(x_k)-g^p_k&=g'(x_k)-\Big(DCT^{(4)}\big(\sqrt{n}\underline{g}^p_\nu\big)\Big)_k=\\
&=g'(x_k)-\Big(DCT^{(4)}\big(DCT^{(4)}\big(\underline{\tilde{g}}^p\big)\big)\Big)_k=\\
&=g'(x_k)-\tilde{g}^p_k=\mathcal{O}(h^4),\qquad h\to 0,
\end{aligned}\tag{53}$$

and this completes the proof.□

4. The algorithms for numerical differentiation

We describe two algorithms based on the method presented in the previous section: the algorithm *FOD* computes the first order numerical derivative of a function by knowing its values in equally spaced points of a closed interval $[a,b]$; the

Table 1
The absolute errors e_f and e_l at the extreme points of the computed first derivative.

h	f_1			f_2		
	e_f	e_l	E_∞	e_f	e_l	E_∞
4.00(-2)	6.18(-5)	9.92(-6)	1.20(-6)	1.33(-4)	7.66(-4)	1.07(-5)
2.00(-2)	7.93(-6)	1.12(-6)	7.53(-8)	1.54(-5)	9.92(-5)	6.69(-7)
1.00(-2)	9.98(-7)	1.32(-7)	4.71(-9)	1.84(-6)	1.26(-5)	4.18(-8)
5.00(-3)	1.24(-7)	1.61(-8)	2.94(-10)	2.26(-7)	1.58(-6)	2.62(-9)
2.50(-3)	1.56(-8)	1.98(-9)	1.85(-11)	2.80(-8)	1.98(-7)	1.64(-10)
1.25(-3)	1.95(-9)	2.45(-10)	1.38(-12)	3.48(-9)	2.48(-8)	1.07(-11)
6.25(-4)	2.44(-10)	3.07(-11)	4.01(-13)	4.34(-10)	3.10(-9)	3.04(-12)

algorithm *NOD* computes the numerical derivative of order $\nu \geq 1$ of a function defined on $[a, b]$. In particular, the algorithm *NOD*, on the basis of the discussion of Section 2.3, iteratively applies the algorithm *FOD*.

The first algorithm is based on Theorem 3.2 and computes the first derivative of a function $f(x)$, $x \in [a, b]$. It requires the knowledge of $f_j = f(a + \xi_j)$, $j = 0, 1, \ldots, n$, where $h = (b - a)/n$, and computes the approximations of $f'(a + x_j)$, $j = 0, 1, \ldots, n - 1$.

For later convenience, we define the following sequences. Given $n, h, \nu > 0$, we define

$$x_k^{(i)} = h\left(k + \frac{i}{2}\right), \quad k = 0, 1, \cdots, n - i, \ i = 1, 2, \cdots, \nu, \tag{54}$$

$$\xi_j^{(i)} = h\left(j + \frac{i-1}{2}\right), \quad j = 0, 1, \cdots, n - i + 1, \ i = 1, 2, \cdots, \nu, \tag{55}$$

we note that $x_k^{(1)} = x_k$ and $\xi_j^{(1)} = \xi_j$.

The second algorithm computes the ν-order derivative of a function f on $[a, b]$. In particular, from the knowledge of $f_j = f(a + \xi_j)$, $j = 0, 1, \ldots, n$, $n > 0$, $h = (b - a)/n$, it computes the approximations of $f^{(\nu)}\left(a + x_{k+d(\nu-1)}^{(\nu)}\right)$, $k = 0, 1, \ldots, m - 1$, where $m = n - \nu - 2d(\nu - 1) + 1$ and d is a non-negative integer defining the number of boundary values of the intermediate derivatives that are not used in the computation of the next order derivative. Indeed, we have found that the derivatives at the boundary points have a slightly higher error than the ones in the interior points, so the choice $d \approx 1, 2$, avoids the propagation of such errors obtained at the boundary points. Note that, the numerical experiments reported in the next section give an evidence of this fact.

In particular, when $d = 0$ this algorithm computes the approximations $D_k^{(\nu)} \approx f^{(\nu)}\left(a + x_k^{(\nu)}\right)$, $k = 0, 1, \ldots, n - \nu$.

5. Numerical results

We describe the results obtained in a numerical experiment with Algorithm *NOD*, where we have considered the following functions

$$f_1(x) = \frac{1}{1 + x^2}, \qquad x \in [0, 1], \tag{56}$$

$$f_2(x) = \cos\left((1 + x)^2\right), \qquad x \in [0, 1], \tag{57}$$

$$f_3(x) = (x^3 - 1)e^x \sin x \cos(x - 3)\cos(x^2 + 2x + 1), \qquad x \in [0, 1]. \tag{58}$$

Their derivatives of order $\nu = 1, 2, 5, 6$ have been computed by using a FORTRAN implementation of Algorithm *NOD*, with different choices of $h = 1/n$, $n = 25, 50, 100,$ 200,400,800,1600, and $d = 1$. The results have been compared with the ones obtained by an extension of the Neville Algorithm [26] implemented in the routine D04AAF of the NAG Library [23]. In particular, f_1 and f_2 have been used to compare the accuracy of these algorithms, instead f_1 and f_3 to compare their computational cost.

The NAG Fortran Compiler Release 6.2 [27] has been used and the experiment has been performed in a Workstation equipped with an Intel(R) Xeon(R) CPU E5-2620 v3 @2.40GHz, operative system Red Hat Enterprise Linux, release 7.5.

The numerical results are reported inTables 1–12, where $x(e)$ denotes $x \cdot 10^e \in \mathbb{R}$. In these tables, we have considered E_2 the mean squared error, E_r the 2-norm relative error and E_∞ the infinity norm error. These errors have been computed by using the computed numerical derivatives and the corresponding exact values of the derivatives, but the values at the two boundary points are not considered in this computation, because Algorithms *FOD* and *NOD* have a bad performance at these extreme points.

Table 2
The numerical test for the rate of convergence of the method.

h	f_1 E_2/h^4	E_r/h^4	E_∞/h^4	f_2 E_2/h^4	E_r/h^4	E_∞/h^4
4.00(-2)	2.55(-1)	4.73(-1)	4.69(-1)	2.49(+0)	1.25(+0)	4.18(+0)
2.00(-2)	2.51(-1)	4.69(-1)	4.71(-1)	2.46(+0)	1.23(+0)	4.18(+0)
1.00(-2)	2.48(-1)	4.67(-1)	4.71(-1)	2.44(+0)	1.21(+0)	4.18(+0)
5.00(-3)	2.47(-1)	4.66(-1)	4.70(-1)	2.43(+0)	1.20(+0)	4.19(+0)
2.50(-3)	2.47(+0)	4.66(-1)	4.73(-1)	2.43(+0)	1.20(+0)	4.20(+0)
1.25(-3)	2.49(-1)	4.71(-1)	5.65(-1)	2.43(+0)	1.20(+0)	4.38(+0)
6.25(-4)	1.39(+0)	2.63(+0)	4.82(+0)	5.90(+0)	2.92(+0)	2.00(+1)

Table 3
The errors with respect to h in the computation of the first derivative of f_1.

h	*NOD* E_2	*D04AAF* E_2	*NOD* E_r	*D04AAF* E_r	*NOD* E_∞	*D04AAF* E_∞
4.00(-2)	6.52(-7)	1.87(-7)	1.21(-6)	3.47(-7)	1.20(-6)	4.51(-7)
2.00(-2)	4.01(-8)	2.74(-11)	7.50(-8)	5.12(-11)	7.53(-8)	7.16(-11)
1.00(-2)	2.48(-9)	5.44(-15)	4.67(-9)	1.02(-14)	4.71(-9)	2.09(-14)
5.00(-3)	1.54(-10)	5.95(-15)	2.91(-10)	1.12(-14)	2.94(-10)	1.80(-14)
2.50(-3)	9.63(-11)	1.19(-14)	1.82(-11)	2.24(-14)	1.85(-11)	4.77(-14)
1.25(-3)	6.09(-13)	1.85(-14)	1.15(-12)	3.49(-14)	1.38(-12)	1.12(-13)
6.25(-4)	2.12(-13)	3.78(-14)	4.01(-13)	7.14(-14)	7.35(-13)	1.83(-13)

Table 4
The errors with respect to h in the computation of the second derivative of f_1.

h	*NOD* E_2	*D04AAF* E_2	*NOD* E_r	*D04AAF* E_r	*NOD* E_∞	*D04AAF* E_∞
4.00(-2)	5.84(-6)	3.10(-7)	6.58(-6)	3.50(-7)	1.10(-5)	7.58(-7)
2.00(-2)	4.07(-7)	4.51(-11)	4.25(-7)	4.70(-11)	9.73(-7)	1.18(-10)
1.00(-2)	2.69(-8)	2.66(-13)	2.72(-8)	2.68(-13)	6.58(-8)	5.84(-13)
5.00(-3)	1.73(-9)	1.05(-12)	1.72(-9)	1.04(-12)	4.18(-9)	2.92(-12)
2.50(-3)	1.17(-10)	3.51(-12)	1.15(-10)	3.47(-12)	3.82(-10)	1.36(-11)
1.25(-3)	1.48(-10)	1.46(-11)	1.46(-10)	1.44(-11)	5.22(-10)	5.91(-11)
6.25(-4)	6.87(-10)	4.00(-11)	6.75(-10)	3.93(-11)	2.12(-9)	1.90(-10)

Table 5
The errors with respect to h in the computation of the derivative of order 5 of f_1.

h	*NOD* E_2	*D04AAF* E_2	*NOD* E_r	*D04AAF* E_r	*NOD* E_∞	*D04AAF* E_∞
4.00(-2)	4.88(-3)	2.70(-2)	1.05(-4)	5.79(-4)	6.64(-2)	4.91(-2)
2.00(-2)	4.81(-4)	5.40(-5)	8.45(-6)	9.50(-7)	1.21(-3)	1.39(-4)
1.00(-2)	4.06(-5)	2.95(-7)	7.28(-7)	5.28(-9)	1.25(-4)	9.17(-7)
5.00(-3)	7.57(-4)	2.34(-6)	1.40(-5)	4.31(-8)	2.03(-3)	7.67(-6)
2.50(-3)	2.61(-2)	8.39(-5)	4.89(-4)	1.57(-6)	6.32(-2)	5.12(-4)
1.25(-3)	7.47(-1)	2.09(-3)	1.41(-2)	3.94(-5)	2.93(0)	1.75(-2)
6.25(-4)	2.99(1)	9.55(-3)	5.68(-1)	1.81(-4)	1.01(+2)	6.59(-2)

The behaviour of Algorithm *FOD* at the extreme points can be evaluated in Table 1. This table reports the infinity error at the inner points E_∞ (also given in the other tables) and the values of the absolute errors e_f and e_l at the extreme points x_0 and x_{n-1}, respectively, only for the first derivatives of f_1 and f_2, computed with Algorithm *FOD*. Of course, similar problems also occur with Algorithm *NOD* for higher order derivatives.

In Table 2 we have reported the ratio of the errors to h^4 for the functions f_i, $i = 1, 2$. These results confirm the rate of convergence $\mathcal{O}(h^4)$ of the proposed method, given in Theorem 3.2, when h is taken in an appropriate interval.

From the Tables 3–10 we can observe that both the routines D04AAF and *NOD* give satisfactory results, even if D04AAF performs slightly better than Algorithm *NOD* except for the function f_1 with $\nu = 5, 6$ and $h = 0.04$. However, this slight higher accuracy of the NAG routine may be explained by the information used in the computation, indeed *NOD* uses only the tabulated points with step size h, while D04AAF uses 21 function values around each one of the tabulated points. Moreover, this preliminary version of the *NOD* implementation does not consider any adaptation strategy such as for instance an optimal step size procedure based on an error evaluation method.

The computational costs of the proposed algorithm *NOD* and of the NAG routine D04AAF are reported in Tables 11–12, where we have considered only the first derivative of functions f_1 and f_3. From these tables we can see that for a high

Table 6
The errors with respect to h in the computation of the derivative of order 6 of f_1.

h	*NOD* E_2	*D04AAF* E_2	*NOD* E_r	*D04AAF* E_r	*NOD* E_∞	*D04AAF* E_∞
4.00(-2)	5.24(-2)	1.18(-1)	1.72(-4)	3.85(-4)	1.05(-1)	1.91(-1)
2.00(-2)	5.24(-3)	3.57(-4)	2.23(-5)	1.52(-6)	1.07(-2)	9.88(-4)
1.00(-2)	5.26(-3)	9.38(-5)	2.09(-5)	3.73(-7)	1.25(-2)	4.03(-4)
5.00(-3)	3.41(-1)	3.28(-3)	1.23(-3)	1.19(-5)	8.58(-1)	2.07(-2)
2.50(-3)	2.34(+1)	3.25(-1)	8.08(+0)	1.12(-3)	5.79(+1)	1.82(+0)
1.25(-3)	1.33(+3)	3.50(-1)	4.48(+0)	1.18(-3)	5.39(+3)	2.77(+0)
6.25(-4)	1.08(+5)	1.77(+1)	3.60(+2)	5.93(-2)	3.61(+5)	9.01(+1)

Table 7
The errors with respect to h in the computation of the first derivative of f_2.

h	*NOD* E_2	*D04AAF* E_2	*NOD* E_r	*D04AAF* E_r	*NOD* E_∞	*D04AAF* E_∞
4.00(-2)	6.38(-6)	3.96(-8)	3.19(-6)	1.98(-8)	1.07(-5)	1.20(-7)
2.00(-2)	3.94(-7)	2.12(-12)	1.96(-7)	1.05(-12)	6.69(-7)	7.26(-12)
1.00(-2)	2.44(-8)	6.25(-15)	1.21(-8)	3.10(-15)	4.18(-8)	2.93(-14)
5.00(-3)	1.52(-9)	1.45(-14)	7.52(-10)	7.15(-15)	2.62(-9)	6.17(-14)
2.50(-3)	9.48(-11)	3.44(-14)	4.68(-11)	1.70(-14)	1.64(-10)	9.64(-14)
1.25(-3)	5.93(-12)	5.46(-14)	2.93(-12)	2.69(-14)	1.07(-11)	2.56(-13)
6.25(-4)	9.01(-13)	1.23(-13)	4.45(-13)	6.07(-14)	3.05(-12)	5.32(-13)

Table 8
The errors with respect to h in the computation of the second derivative of f_2.

h	*NOD* E_2	*D04AAF* E_2	*NOD* E_r	*D04AAF* E_r	*NOD* E_∞	*D04AAF* E_∞
4.00(-2)	4.02(-5)	2.79(-8)	5.24(-6)	3.64(-9)	6.04(-5)	6.91(-8)
2.00(-2)	2.77(-6)	2.12(-12)	3.48(-7)	2.67(-13)	6.69(-6)	5.36(-12)
1.00(-2)	1.86(-7)	1.22(-12)	2.31(-8)	1.52(-13)	5.15(-7)	5.43(-12)
5.00(-3)	1.21(-8)	3.49(-12)	1.50(-9)	4.31(-13)	3.52(-8)	9.21(-12)
2.50(-3)	7.84(-10)	9.37(-12)	9.69(-10)	1.16(-12)	2.42(-9)	3.31(-11)
1.25(-3)	5.54(-10)	4.34(-11)	6.84(-11)	5.36(-12)	1.69(-9)	2.18(-10)
6.25(-4)	2.78(-9)	1.06(-10)	3.43(-10)	1.31(-11)	7.98(-9)	6.17(-10)

Table 9
The errors with respect to h in the computation of the derivative of order 5 of f_2.

h	*NOD* E_2	*D04AAF* E_2	*NOD* E_r	*D04AAF* E_r	*NOD* E_∞	*D04AAF* E_∞
4.00(-2)	1.52(-2)	4.69(-3)	3.14(-5)	9.70(-6)	1.90(-2)	1.32(-2)
2.00(-2)	8.36(-4)	2.90(-6)	1.54(-6)	5.35(-9)	1.52(-3)	6.76(-6)
1.00(-2)	1.16(-4)	1.65(-7)	2.14(-7)	3.05(-10)	3.63(-4)	8.58(-7)
5.00(-3)	2.83(-3)	9.88(-6)	5.33(-6)	1.86(-8)	8.61(-3)	3.98(-5)
2.50(-3)	7.43(-2)	2.21(-4)	1.42(-4)	4.22(-7)	1.95(-1)	6.93(-4)
1.25(-3)	2.96(+0)	3.29(-3)	5.69(-3)	6.33(-6)	8.58(+0)	3.66(-2)
6.25(-4)	1.27(+2)	5.91(-2)	2.44(-1)	1.14(-4)	3.25(+2)	1.65(-1)

number of derivative computations the proposed algorithm has a lower computational cost than Neville's one. The different computational efficiency is more evident when the function tabulation has a high cost, as in the case of f_3.

Ultimately, the proposed algorithm shows similar performance of D04AAF routine, despite this NAG routine is robust, documented and well tested. These results bode well for the development of a scientific software that outperforms also the efficient routines of NAG library.

The proposed procedure can easily be generalised to functions $F : [0, 1]^s \to \mathbb{R}$, with $s \geq 2$. For example, for $s = 2$ the input is a matrix $\underline{F} \in \mathbb{R}^{(n+1)\times(n+1)}$ containing the values of F at (ξ_i, ξ_j), $i, j = 0, 1, \ldots, n$, and using FOD with input $a = 0$, $b = 1$ and $\underline{f} = \underline{F}(:, j)$, we obtain $\underline{D}$ whose components are approximations of $F'_x(x_i, \xi_j)$, $i = 0, 1, \ldots, n-1$. Similar considerations can be done for other derivatives or an higher dimension of the domain of F. So that, in order to show the performances of our proposed method when $s \geq 2$, we chose $s = 2$ and we reported in Table 13 the infinity norm of errors obtained by using this generalisation to compute, F'_x, F'_y and F''_{yy} where $F(x, y) = f_2(xy + x)$. As in one-dimensional case, these errors are computed only in the inner nodes and the results have the same behaviours observed for functions with one input. Hence the number of inputs of a function does not affect the accuracy of the calculated derivatives.

Table 10
The errors with respect to h in the computation of the derivative of order 6 of f_2.

h	*NOD* E_2	*D04AAF* E_2	*NOD* E_r	*D04AAF* E_r	*NOD* E_∞	*D04AAF* E_∞
4.00(-2)	5.34(-2)	4.12(-3)	2.46(-5)	1.90(-6)	8.88(-2)	6.51(-3)
2.00(-2)	8.18(-3)	1.29(-5)	4.78(-6)	7.57(-9)	2.09(-2)	3.67(-5)
1.00(-2)	2.10(-2)	4.64(-5)	1.22(-5)	2.71(-8)	6.53(-2)	2.20(-4)
5.00(-3)	1.29(+0)	1.07(-2)	6.87(-4)	5.69(-6)	3.95(+0)	7.73(-2)
2.50(-3)	6.65(+1)	5.23(-1)	3.31(-2)	2.60(-4)	1.67(+2)	3.26(+0)
1.25(-3)	5.31(+3)	9.33(-1)	2.55(+0)	4.47(-4)	1.55(+4)	2.66(+0)
6.25(-4)	4.62(+5)	4.76(+1)	2.17(+2)	2.24(-2)	1.16(+6)	1.75(+2)

Table 11
The computational cost, in seconds, for computing the first derivative of f_1 with *FOD* and D04AAF.

h	*FOD*	*D04AAF*
4.00(-2)	1.01(-3)	3.30(-5)
2.00(-2)	2.66(-4)	5.60(-5)
1.00(-2)	2.82(-4)	7.50(-5)
5.00(-3)	4.01(-4)	1.90(-4)
2.50(-3)	4.23(-4)	3.08(-4)
1.25(-3)	5.70(-4)	8.16(-4)
6.25(-4)	8.24(-4)	1.24(-3)

Table 12
The computational cost, in seconds, for computing the first derivative of f_3 with *FOD* and D04AAF.

h	*FOD*	*D04AAF*
4.00(-2)	2.60(-4)	1.01(-4)
2.00(-2)	2.76(-4)	1.72(-4)
1.00(-2)	2.93(-4)	3.18(-4)
5.00(-3)	4.23(-4)	7.83(-4)
2.50(-3)	4.52(-4)	1.22(-3)
1.25(-3)	6.52(-4)	2.44(-3)
6.25(-4)	1.10(-3)	5.10(-3)

Table 13
The infinity norm of errors with respect to h in the computation of F'_x, F'_y and F''_{yy} where $F(x,y) = f_2(xy + x)$.

h	*FOD* F'_x	*FOD* F'_y	*NOD* F''_{yy}
4.00(-2)	1.07(-5)	3.35(-7)	1.67(-6)
2.00(-2)	6.69(-7)	2.09(-8)	1.29(-7)
1.00(-2)	4.18((-8)	1.31(-9)	8.79(-9)
5.00(-3)	2.62(-9)	8.17(-11)	5.82(-10)
2.50(-3)	1.63(-10)	5.19(-12)	3.37(-10)
1.25(-3)	1.07(-11)	7.11(-13)	1.10(-9)
6.25(-4)	3.05(-12)	1.93(-12)	5.59(-9)

Finally, in order to numerically verify the robustness of the proposed method, we added to the input vector $\underline{f}$ a random error $\underline{\epsilon}(\delta)$ whose components were uniformly distributed in $[-\delta, \delta]$ with $\delta = 10^e$, $e = -14, -13, \ldots, -1$. We computed, with FOD, the first derivative and the ratio between the relative error of the computed derivative and the relative error of the data, with respect to the infinity norm, we repeated this procedure 10 times and we denoted with $K(\delta)$ the means of these ratios. In the vertical axes of Fig. 1, it is reported the values $K(\delta)$ obtained with the above described procedure, when we use FOD with $h = 1.25 \cdot 10^{-3}$ to numerically compute f_1' when the data are contaminated with a reference error level $\delta = 10^e$, and $e = -14, -13, \ldots, -1$. From this graph we can say that the conditional number of the proposed algorithm is approximated by 2704 (equal to the mean of the values $K(10^e)$ represented in Fig. 1). For different choices of f and of h we obtained similar behaviours, in particular, the computed conditional number when $h = 4 \cdot 10^{-2}$ is 70.8. These numerical tests prove the robustness of the proposed algorithm.

6. Conclusions

We have proposed a method to compute the numerical derivatives of a function known at equispaced points. The proposed method uses the FFT and the singular value expansion of the Volterra integral operator associated to the ν-derivative operator. The use of the FFT is justified by the particular form of the singular functions. Two algorithms based on the pro-

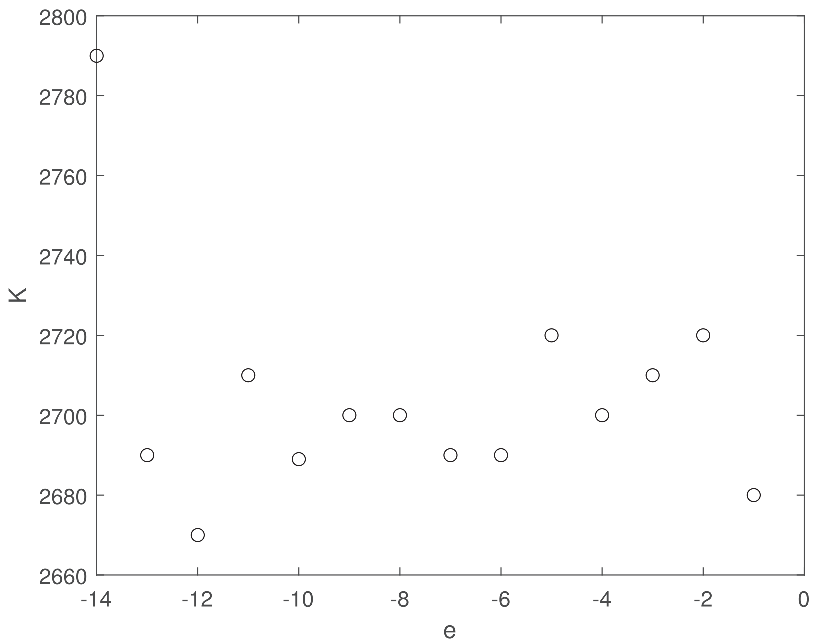

Fig. 1. The mean values $K(\delta)$ obtained with FOD with $h = 1.25 \cdot 10^{-3}$ in the computation of f_1' when the data are contaminated with a reference error level $\delta = 10^e$, the exponent e is represented in the horizontal axes.

Algorithm 1 (First order derivative) $\mathbf{FOD}\big(a, b, n, \underline{f}, \underline{D}\big)$ Given $n \in \mathbb{N}$, $n > 0$, $h = (b-a)/n$, and $\underline{f} = (f_0, f_1, \ldots, f_n) \in \mathbb{R}^{n+1}$, where $f_j = f(a + \xi_j)$, $j = 0, 1, \ldots, n$, compute $\underline{D} = (D_0, D_1, \ldots, D_{n-1}) \in \mathbb{R}^n$, containing the approximation $D_j \approx f'(a + x_j)$, $j = 0, 1, \ldots, n-1$.

Construct the vector $\underline{g} \in \mathbb{R}^n$, where its components are $g_j = f_j - f_0$, $j = 1, \ldots, n$
for $l = 0, \ldots, n-1$ **do**
  Compute the quantity $g^p_{l,\nu}$ by using formula (43)
end for
for $k = 0, \ldots, n-1$ **do**
  Compute g^p_k by using formula (42)
  Compute $D_k = \frac{g^p_k}{b-a}$
end for
return $\underline{D}$

Algorithm 2 (ν-order derivative) $\mathbf{NOD}\big(a, b, n, d, \underline{f}, \underline{D}^{(\nu)}, \nu\big)$ Given $n \in \mathbb{N}$, $n > 0$, $h = (b-a)/n$, $m = n - \nu - 2d(\nu - 1) + 1$, and $\underline{f} = (f_0, f_1, \ldots, f_n) \in \mathbb{R}^{n+1}$, where $f_j = f(a + \xi_j)$, $j = 0, 1, \ldots, n$, compute $\underline{D}^{(\nu)} = (D_0, D_1, \ldots, D_{m-1}) \in \mathbb{R}^m$, containing the approximations $D_k \approx f^{(\nu)}\left(a + x^{(\nu)}_{k+d(\nu-1)}\right)$, $k = 0, 1, \ldots, m-1$.

Let $\underline{D}^{(0)} = \underline{f} \in \mathbb{R}^{n+1}$
for $l = 1, 2, \ldots, \nu$ **do**
  $m = n - l + 1;$
  Compute $\underline{D}^{(l)} \in \mathbb{R}^m$ by FOD$\left(\xi^{(l)}_{d(l-1)}, \xi^{(l)}_{m+d(l-1)}, m, \underline{D}^{(l-1)}, \underline{D}^{(l)}\right)$.
  if $l < \nu$ **then**
    delete from $\underline{D}^{(l)}$ the first d components and the last$d$ components
    set $n = n - 2d$
  end if
end for
return $\underline{D}^{(\nu)}$

posed method have been given and implemented for functions f defined on a closed interval $[a, b]$: the algorithm *FOD* to compute the first derivative f'; the algorithm *NOD* to compute the derivative $f^{(\nu)}$ of order $\nu \geq 1$. The rate of convergence of the method has been proved and numerically validated. The numerical results obtained with the proposed method have been compared with the ones obtained by an extension of the Neville Algorithm implemented in the routine D04AAF of the NAG library. Despite the routine D04AAF is a robust, documented and tested numerical algorithm, the proposed algorithm has shown similar performance. Moreover, when the derivatives are required for a high number of points, the proposed algorithm has a lower computational cost than the one of D04AAF.

The promising results obtained in this paper motivate further studies of the proposed method to obtain a state of the art procedure for the numerical differentiation. Other interesting problems are the application of this method to the solution of differential equations, the numerical differentiation from scattered and/or multivariate data as well as the stability analysis and the corresponding stabilization techniques to deal with noisy function values. Moreover, the main improvements of the method can arise from: higher integration formulas than formula (43); an automatic selection of the optimal step size h to use in the computation; the joint use of the operator $\mathcal{K}$ in (1)-(2) and of its adjoint operator. Further improvements relate to the refinement of the algorithm for the first order derivative and a direct use of the SVE, of the corresponding operator, for the computation of the derivatives of order $\nu > 1$.

Data availability

Data will be made available on request.

References

[1] B. Yin, Y. Ye, Recovering the local volatility in Black-Scholes model by numerical differentiation, Appl. Anal. 85 (6–7) (2006) 681–692.
[2] X. Wan, Y. Wang, M. Yamamoto, Detection of irregular points by regularization in numerical differentiation and application to edge detection, Inverse Probl. 22 (3) (2006) 1089–1103.
[3] J. Cheng, X.Z. Jia, Y.B. Wang, Numerical differentiation and its applications, Inverse Probl. Sci. Eng. 15 (4) (2007) 339–357.
[4] N. Egidi, P. Maponi, A spectral method for the solution of boundary value problems, Appl. Math. Comput. 409 (2021).
[5] N. Egidi, P. Maponi, An SVE approach for the numerical solution of ordinary differential equations, in: Y. Sergeyev, D. Kvasov (Eds.), Numerical Computations: Theory and Algorithms. NUMTA 2019, Lecture Notes in Computer Science, Vol. 11973, Springer, Cham, 2020, pp. 70–85.
[6] R.S. Anderssen, M. Hegland, For numerical differentiation, dimensionality can be a blessing!, Math. Comput. 68 (227) (1999) 1121–1141.
[7] J.N. Lyness, C.B. Moler, Van der Monde systems and numerical differentiation, Numer. Math. 8 (1966) 458–464.
[8] O. Davydov, R. Schaback, Minimal numerical differentiation formulas, Numer. Math. 140 (3) (2018) 555–592.
[9] A.G. Ramm, A.B. Smirnova, On stable numerical differentiation, Math. Comput. 70 (235) (2001) 1131–1153.
[10] O. Davydov, R. Schaback, Error bounds for kernel-based numerical differentiation, Numer. Math. 132 (2) (2016) 243–269.
[11] T.J. Rivlin, Optimally stable Lagrangian numerical differentiation, SIAM J. Numer. Anal. 12 (5) (1975) 712–725.
[12] T. Tsuda, Numerical differentiation of functions of very many variables, Numer. Math. 18 (1971) 327–335.
[13] R.S. Anderssen, P. Bloomfield, Numerical differentiation procedures for non-exact data, Numer.. Math. 22 (1974) 157–182.
[14] I. Knowles, R. Wallace, A variational method for numerical differentiation, Numer. Math. 70 (1) (1995) 91–110.
[15] S. Lu, S.V. Pereverzev, Numerical differentiation from a viewpoint of regularization theory, Math. Comput. 75 (256) (2006) 1853–1870.
[16] Y.B. Wang, X.Z. Jia, J. Cheng, A numerical differentiation method and its application to reconstruction of discontinuity, Inverse Probl. 18 (6) (2002) 1461–1476.
[17] F.S.V. Bazán, L. Bedin, Filtered spectral differentiation method for numerical differentiation of periodic functions with application to heat flux estimation, Comput. Appl. Math. 38 (4) (2019) 165–188.
[18] Z. Zhao, Z. Meng, G. He, A new approach to numerical differentiation, J. Comput. Appl. Math. 232 (2009) 227–239.
[19] Z. Zhao, A Hermite extension method for numerical differentiation, Appl. Numer. Math. 159 (2021) 46–60.
[20] Y. Luo, Galerkin method with trigonometric basis on stable numerical differentiation, Appl. Math. Comput. 370 (2020) 124912.
[21] Z. Zhao, L. You, A numerical differentiation method based on Legendre expansion with super order Tikhonov regularization, Appl. Math. Comput. 393 (2021) 125811.
[22] X. Huang, C. Wu, J. Zhou, Numerical differentiation by integration, Math. Comput. 83 (286) (2014) 789–807.
[23] The NAG Library, The Numerical Algorithms Group (NAG), Oxford, United Kingdom, 2022.
[24] H. Zhang, Q. Zhang, Sparse discretization matrices for Volterra integral operators with applications to numerical differentiation, J. Integr. Eqs. Appl. 23 (1) (2011) 137–156.
[25] N. Egidi, P. Maponi, The singular value expansion of the Volterra integral equation associated to a numerical differentiation problem, J. Math. Anal. Appl. 460 (2018) 656–681.
[26] J.N. Lyness, C.B. Moler, Generalised Romberg methods for integrals of derivatives, Numer. Math. 14 (1969) 1–13.
[27] The NAG Fortran CompilerT, Oxford, United Kingdom, 2022. https://www.nag.com/content/nag-fortran-compiler-release-62