Received: 14 June 2019 | Revised: 24 April 2020 | Accepted: 11 May 2020

DOI: 10.1002/nla.2314

SPECIAL ISSUE PAPER

WILEY

PARASTIELTJES: Parallel computation of Gauss quadrature rules using a PARAREAL-like approach for the Stieltjes procedure

Martin J. Gander | **Thibaut Lunet**

Section of Mathematics, University of Geneva, Canton of Geneva, Switzerland

Correspondence
Thibaut Lunet, 2-4 rue du Lièvre, 1211 Genève 4, Switzerland.
Email: thibaut.lunet@unige.ch

Summary

The computation of Gauss quadrature rules for arbitrary weight functions using the Stieltjes algorithm is a purely sequential process, and the computational cost significantly increases when high accuracy is required. PARASTIELTJES is a new algorithm to compute the recurrence coefficients of the associated orthogonal polynomials in parallel, from which the nodes and weights of the quadrature rule can then be obtained. PARASTIELTJES is based on the time-parallel PARAREAL algorithm for solving time-dependent problems, and thus enlarges the applicability of this time parallel technique to a further, new area of scientific computing. We study PARASTIELTJES numerically for different weight functions, and show that substantial theoretical speedup can be obtained when high accuracy is needed. We also present an asymptotic approximation for the node and weight distribution of Gauss quadrature rules, which can be used effectively in PARASTIELTJES.

KEYWORDS

discretization method, Gauss quadrature rule, orthogonal polynomials, parallel computing, PARAREAL, Stieltjes procedure

1 | INTRODUCTION

Gauss quadrature rules are very powerful techniques for the numerical approximation of integrals. They can be adapted to integrals with very general weight functions $w(t)$,

$$\int_a^b f(t)w(t)dt, \tag{1}$$

and lead to the highest possible accuracy for a given number of function evaluations. In addition to classical weight functions $w(t)$, numerous applications leading to specific $w(t)$ can be found in the literature (eg, inverse Laplace transforms[1] or radiative transfer[2]). To determine a Gauss quadrature rule for a given weight function, one can use the Stieltjes procedure[1], a well-known sequential method to compute the associated orthogonal polynomials, from which the nodes and weights of the associated Gauss quadrature rule can then be obtained. To use the Stieltjes procedure on the computer, Gautschi introduced the discretization method[3] that allows to compute such general Gauss

[1]Called Lanczos in linear algebra

quadrature rules to a desired degree of accuracy. However, the more accuracy one needs, the higher the computational cost becomes for the discretized Stieltjes procedure, and speeding up the computation using multiple processors seems not possible because of the inherent sequential nature of the Stieltjes procedure.

On the other hand, substantial research efforts have been devoted over the last two decades to invent and study time-parallel methods for solving evolution problems governed by ordinary or partial differential equations. These methods try to break the inherent sequential nature of evolution problems by decomposing them in the time direction into several subproblems, on which computations can then be performed in parallel, even though the solution is not yet completely determined on the previous subproblems. Many different strategies have been developed, for a detailed review.[4] These methods have found their use in numerous applications, for example, power systems with MGRIT,[5] optimal control with PFASST,[6] or direct numerical simulations of turbulent flows with PARAREAL.[7] Recently some applications have even been considered for problems without any time dependence, like the training of neural networks.[8]

We propose here a new such application: the computation of orthogonal polynomials using ideas from the PARAREAL algorithm. The PARAREAL algorithm was invented in Reference 9, and sparked renewed and substantial interest in time parallel methods; for convergence analyses, see References 10,11. With the PARAREAL algorithm, one introduces time parallelization into already existing time-dependent problem solvers, as black box solver, manipulating their inputs and outputs to obtain different accuracies. With the PARAREAL approach, one does not really need to be solving a time-dependent problem, any recurrence relation can be parallelized if different degrees of accuracy can be chosen. The discretization method based on the Stieltjes procedure thus fits the PARAREAL paradigm. However, the performance of PARAREAL depends strongly on the type of problem and solver, for example, there are known difficulties for the solution of hyperbolic PDEs.[12] Hence any application of this algorithm to a new type of problem needs to be carefully studied. This is the goal of this article, where we design a new method combining PARAREAL with the discretization method and the Stieltjes procedure, to obtain PARASTIELTJES. PARASTIELTJES is the main contribution of this article, together with thorough numerical tests to evaluate its accuracy and efficiency. A second important contribution is the concept of asymptotic approximation of the Gauss quadrature nodes and weights, based on some interesting properties of Gauss quadrature rules that may not be well known in the literature, which are very useful in the scope of the PARASTIELTJES algorithm.

Our article is structured as follows: in Section 2, we give a general description of Gauss quadrature rules and of the discretization method, along with the description of the asymptotic approximation. Then in Section 3 we present the PARAREAL algorithm and introduce the new PARASTIELTJES algorithm. In Section 4, we show numerical experiments to evaluate the performance and efficiency of PARASTIELTJES, and present a first analysis of the influence of the main parameters on PARASTIELTJES. Finally, in Section 5, we discuss a further potential application and implementation considerations for our new method. We give a summary and an outlook on future work in the concluding Section 6.

2 | COMPUTATION OF GAUSS QUADRATURE RULES

2.1 | Orthogonal polynomials and the Stieltjes procedure

We consider an n-point Gauss quadrature rule applied to a real-valued scalar function f,

$$\int_a^b f(t)w(t)dt = \sum_{j=1}^{J} \omega_j f(\tau_j) + R_J(f), \tag{2}$$

where $-\infty \leq a < b \leq \infty$, $w : \mathbb{R} \mapsto \mathbb{R}_+^*$ [2] is a strictly positive weight function such that all its moments $\int_a^b t^m w(t)dt$ exist for all positive integers m, and the remainder $R_J(p)$ vanishes for any polynomials p of degree $2J-1$ ($p \in \mathbb{P}_{2J-1}$). It is known[13] that the quadrature nodes and weights $(\tau_j,\omega_j)_{1 \leq j \leq J}$ are linked to a particular sequence of monic[3]

[2]We note $\mathbb{R}_+^* = \{x \in \mathbb{R},\ x > 0\}$

[3]A monic polynomial has its coefficient of higher degree equal to 1.

polynomials $(\pi_j)_{1 \le j \le J}$ associated with the weight function w, that are orthogonal with respect to the weighted scalar product

$$\langle p, q \rangle := \int_a^b p(t)q(t)w(t)dt. \tag{3}$$

In particular, the $(\pi_j)_{1 \le j \le J}$ form a basis for $\mathbb{P}_J$, and are fully determined by the three-term recurrence relation

$$\pi_{j+1}(t) = (t - \alpha_j)\pi_j(t) - \beta_j\pi_{j-1}(t) \quad \text{with } \pi_{-1}(t) = 0,\ \pi_0(t) = 1. \tag{4}$$

Here the $(\alpha_j,\beta_j)_{0 \le j \le J-1}$ not only define the orthogonal polynomials, they also allow us to compute the nodes and weights of the quadrature rule: if we denote the Jacobi matrix of the three-term recurrence by

$$\mathcal{J}_\infty := \begin{bmatrix} \alpha_0 & \sqrt{\beta_1} & & \\ \sqrt{\beta_1} & \alpha_1 & \sqrt{\beta_2} & \\ & \sqrt{\beta_2} & \alpha_2 & \sqrt{\beta_3} \\ & & \ddots & \ddots & \ddots \end{bmatrix}, \tag{5}$$

with $\mathcal{J}_J$ its leading principal submatrix of size $J \times J$, then

$$\tau_j = \lambda_j, \quad \omega_j = \beta_0 \boldsymbol{v}_{j,1}^2, \quad \forall j \in \{1, \dots, J\}, \tag{6}$$

where λ_j are the eigenvalues and $\boldsymbol{v}_{i,1}$ are the first components of the eigenvectors of $\mathcal{J}_J$. In practice, one generally uses the Golub-Welsch algorithm[14] to compute the nodes and weights in an efficient manner.[4] The Golub-Welsch algorithm corresponds to a modified QR algorithm where only the first components of the eigenvectors are computed simultaneously with the eigenvalues,[17] saving the cost of computing the complete eigenvectors. Note that if one knows the J first coefficients (α_j,β_j), then one can compute all quadrature rules up to n points, just by applying the Golub-Welsch algorithm to the corresponding leading submatrix of $\mathcal{J}_J$. For computing Gauss quadrature rules, one thus focuses on computing the recurrence coefficients (α_j,β_j), from which the nodes and weights can be easily obtained.

To compute the recurrence coefficients (α_j,β_j), one can use the Stieltjes procedure:[18] suppose that we know how to evaluate the scalar product (3), we can compute the recurrence coefficients from the orthogonal polynomials,

$$\alpha_j = \frac{\langle t\pi_j, \pi_j \rangle}{\langle \pi_j, \pi_j \rangle}, \quad \forall j \in \mathbb{N}, \tag{7}$$

$$\beta_0 = \langle \pi_0, \pi_0 \rangle = \int_a^b w(t)dt, \quad \beta_j = \frac{\langle \pi_j, \pi_j \rangle}{\langle \pi_{j-1}, \pi_{j-1} \rangle}, \quad \forall j \in \mathbb{N}^+. \tag{8}$$

Once the recurrence coefficients are computed up to the jth index, one uses the three-term recurrence relation (4) to compute the $(j + 1)$th orthogonal polynomial, and then continues recursively with the next recurrence coefficients. Like the Lanzcos algorithm, the Stieltjes procedure can be numerically unstable for large values of j[17(sec. 5.1)], due to roundoff errors. But as mentioned in Reference 3, sec. 4, the Stieltjes procedure is still quite efficient when the number of required recurrence coefficients is not too large (less than about 300 in general). It remains to determine a method to compute the scalar product (3), for which the nodes and weights of the associated Gauss quadrature rule are still unknown.

2.2 | The discretization method for nonclassical quadrature rules

An elegant idea to approximately compute the scalar product (3) was originally proposed by Gautschi,[19] as a way to avoid numerical instabilities arising when using the classical methods at that time. The idea is to approximate the continuous scalar product (3) by a discrete one defined by some quadrature rule,

$$\langle p, q \rangle \simeq \sum_{j=1}^{N_q} \omega_j^D p(\tau_j^D)q(\tau_j^D) = \langle p, q \rangle_{N_q}. \tag{9}$$

[4]Note there are also other approaches; for a comparison with Newton Maehly, see Reference 15, secs. 9.3.3 and 9.3.4, and the race is continuing.[16]

One can then compute approximate recurrence coefficients $(\alpha_j^{N_q}, \beta_j^{N_q})_{0\le j\le J-1}$ and orthogonal polynomials $(\pi_j^{N_q})_{1\le j\le J}$ using the discretized Stieltjes procedure. The following theorem justifies the convergence of the discretization method:

Theorem 1 (Convergence of the discretization method). *Let w be a positive weight function on $[-1,1]$, having finite moments of all orders and let $\langle p,q\rangle_{N_q}$ be a discrete scalar product defined as in (9) such that $\tau_j^D(N_q)$ are distinct values in $[-1,1]$ and $\omega_j^D(N_q)>0$ for each N_q. Assume that*

$$\lim_{N_q\to\infty}\langle p,q\rangle_{N_q}=\langle p,q\rangle \tag{10}$$

for any real polynomial p and q. Then, for any fixed j,

$$\left(\alpha_j^{N_q},\beta_j^{N_q}\right)\to_{N_q\to\infty}(\alpha_j,\beta_j), \tag{11}$$

where (α_j,β_j) are the recurrence coefficients of the orthogonal polynomials associated with w.

Remark 1. The proof of Theorem 1 was originally given in Reference 19, sec. 4, but a more detailed version can be found in Reference 13, th. 2.32.

The last step consists in defining the nodes and weights of the quadrature rule used to define the discrete inner product in (9). The idea originally suggested in Reference 19, and again supported in References 3,20 is to use a Fejer-I rule,

$$\int_{-1}^{1} f(t)dt\simeq\sum_{j=1}^{N_q}\omega_j^F f(\tau_j^F). \tag{12}$$

In that case the discrete inner product is obtained by setting

$$\tau_j^D:=\tau_j^F,\quad \omega_j^D:=\omega_j^F w(\tau_j^F). \tag{13}$$

Note that the discrete inner product and Theorem 1 are restricted to integration over $[-1,1]$. Other quadrature rules over unbounded intervals can be reduced to the canonical interval $[-1,1]$ by a suitable change of variables described in Reference 13, sec. 2.2.2. We will however consider only weight functions over $[-1,1]$ here.

2.3 | The discretization method based on an asymptotic approximation

Before investigating the accuracy of the discretization method, we want to introduce an alternative way of constructing a discrete quadrature rule, based on an interesting property of Gauss quadrature rules. Let w be a weight function on $[-1,1]$ and $(\tau_j,\omega_j)_{1\le j\le J}$ be its Jth associated nodes and weights. We use the following definitions:

Definition 1 (Node distribution function). Let δ_J:(0,1) $\to[-1,1]$ be the continuous piecewise linear function defined by its values τ_j at the nodes $\frac{j}{J+1}$,

$$\delta_J\left(\frac{j}{J+1}\right)=\tau_j\quad\forall j\in\{1,\dots,J\}. \tag{14}$$

We call δ_J the node distribution function associated with the weight function w.

Definition 2 (Continuous weight function). Let $\omega_J:\ (-1,1)\to\mathbb{R}_+$ be the continuous piecewise linear function defined by its values ω_j at the nodes τ_j,

$$\omega_J(\tau_j):=\omega_j\quad\forall j\in\{1,\dots,J\}. \tag{15}$$

We call ω_J the continuous weight function associated with the weight function w.

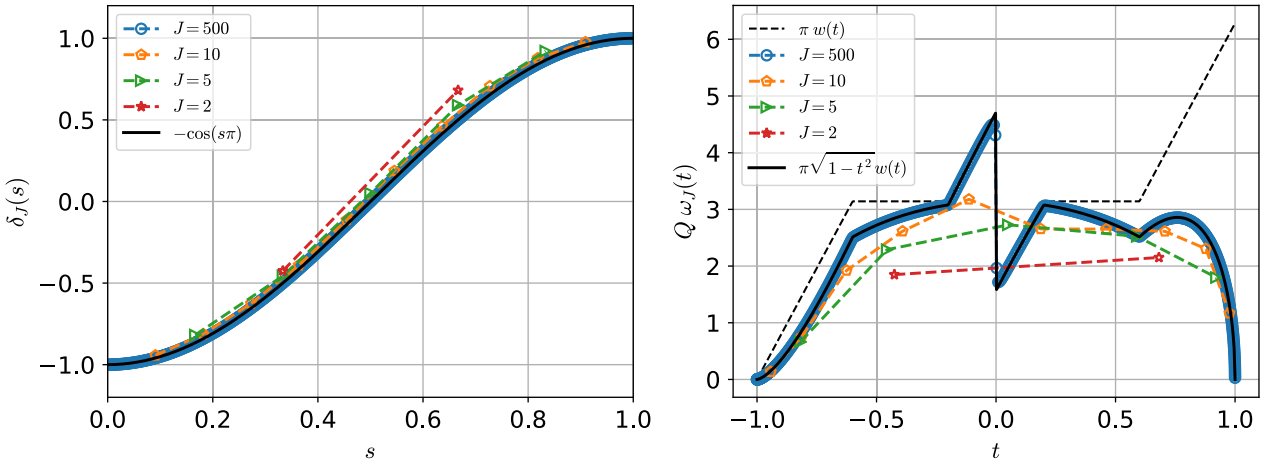

FIGURE 1 Convergence of the node distribution function (left) and continuous weight function (right) for the ELECTRO weight function

Then we have the following theorem:

Theorem 2 (Asymptotic approximation of the Gauss quadrature rule). *For all Szegö-type weight functions defined on* $[-1,1]$ *(ie,* $\int_{-1}^{1} \ln(w(t))/\sqrt{1-t^2}dt > -\infty$*, cf [13, eq. 1.3.10]), we have*

$$\lim_{J\to\infty} \delta_J(s) = -\cos(s\pi) \tag{16}$$

and

$$\lim_{J\to\infty} J\omega_J(t) = \pi\sqrt{1-t^2}w(t). \tag{17}$$

Remark 2. For the convergence of the weights, the proof can be found in the literature in several forms. It is for instance called the "Circle Theorem" by Gautschi,[21] and was proved using the Christoffel functions in Reference [22, th. 4.5.2]. Interestingly, this asymptotic behavior was known already much earlier: Akhiezer[23] gave a similar formula for Chebyshev quadrature rules over [0,1]. For the convergence of the node distribution, it is known since Szegö[24, th. 12.7.2] that it does not depend on the weight function. The result was however never formulated as in (16), but rather using the density function of the zeros (which can be seen as the inverse of the distribution function). However, this mapping of the nodes into [0,1] using the cosine function is present in Szegö[24, th. 12.7.2], and we verified this asymptotic behavior numerically for numerous weight functions. Nevertheless, a complete and formal proof of this node distribution convergence would still be needed, but this is beyond the scope of the present article.

Observation 1. Following (16), the node distribution of a Gauss quadrature rule associated with any Szegö-type weight function defined on $[-1,1]$ tends asymptotically to the node distribution of a Gauss quadrature rule associated with the Chebyshev polynomial of the second kind. Furthermore, (17) indicates that the asymptotic approximation for the weights is exact at each J for Gauss quadrature rules associated with the Chebyshev polynomial of the first kind.

An illustration of the asymptotic approximation is given in Figure 1, using a piecewise linear weight function defined on $[-1,1]$ by

$$w(t) = \frac{5}{2}(t+1)\left(t<\frac{3}{2}\right) + \frac{5}{2}\left(t-\frac{3}{5}\right)\left(t>\frac{3}{5}\right) + \left(t\geq\frac{3}{2}\right) + \frac{5}{2}\left(t+\frac{1}{5}\right)\left(t>-\frac{1}{5}\right)(t<0) + \frac{5}{2}\left(t-\frac{1}{5}\right)\left(t<\frac{1}{5}\right)(t\geq 0), \tag{18}$$

where $(a<b)$ is equal to 0 or 1 whether or not the inequality holds. We call the function defined in (18) the "ELECTRO" weight function, because of its particular shape, represented by the dotted line in Figure 1. We compute the recurrence coefficients using the discretization method with a 10^{-6} accuracy, and we get the nodes and weights using the Golub-Welsch algorithm. We see in Figure 1 the rapid convergence of the nodes and weights to the asymptotic formula given in Theorem 2, despite the discontinuity of the "ELECTRO" weight function at t=0.

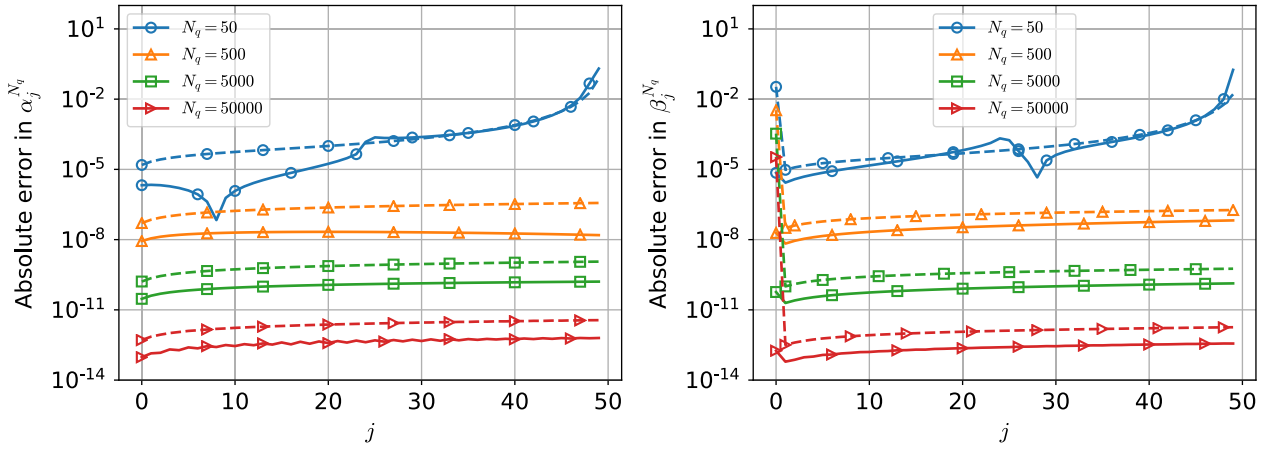

FIGURE 2 Accuracy of the discretization method for the JACOBI weight function, using Fejer-I for the discrete quadrature rule (solid lines) and the asymptotic approximation (dashed lines)

From the asymptotic approximation another procedure to build a discrete inner product for (9) can be derived: one simply takes the asymptotic formulas (16) and (17) to explicitly compute the discrete nodes and weights $(\tau_j^D, \omega_j^D)_{1\le j\le N_q}$, and Theorem 2 ensures the convergence of the discrete scalar product to the one associated with the weight function w when $N_q \to \infty$, which makes the discretization method works due to Theorem 1.

2.4 | Accuracy of the discretization method and motivation for a parallel algorithm

We now focus on the accuracy of the discretization method, depending on the value of N_q. To this end, we choose for the weight function on $[-1,1]$

$$w(t) = \sqrt[4]{1-t}\sqrt{1+t}, \tag{19}$$

which we call JACOBI, as it is associated with a particular case of the Jacobi orthogonal polynomials, for which the recurrence coefficients are known in closed form[13(tab. 1.1)]. Thus we can compute the absolute error of the coefficients $(\alpha_j^{N_q}, \beta_j^{N_q})_{0\le j\le J-1}$ compared with their exact values. The result is shown in Figure 2 for J=50 and increasing values of N_q, and we observe a similar convergence behavior for the recurrence coefficients α and β.

One can also observe that the Fejer-I rule gives in general more accurate results than the asymptotic approximation for a fixed value of N_q, although there is only a ratio of order 10 between the different errors, except for the $\beta_0^{N_q}$ coefficient where the difference is more pronounced. However it should be mentioned that the discrete rule using the asymptotic approximation is obtained explicitly, while the Fejer-I discrete rule needs the computation of fast Fourier transforms, which are computationally more expensive for large values of N_q.

Furthermore, Figure 2 shows that large values of N_q are required to get high accuracy for the recurrence coefficients. Hence the computational cost of the discrete scalar product (9) will notably increase when higher precision is required, making the Stieltjes procedure described in Section 2.1 quite expensive. To illustrate this, reducing the error to machine precision for the J=50 coefficients (N_q~5.0e5) required more than 6 seconds on a recent laptop computer (Intel(R) Core(TM) i7-8565U CPU @ 1.80 GHz), using the optimized NUMPY library for vector operations and sum calculation (version 1.17.4, PYTHON 3.7.6). Going to J=250 coefficients required around half a minute, and numerical experiments considering weight functions with singularities or discontinuities showed a drastic increase in computation time. As this last algorithm is inherently sequential, one can consider the application of a particular parallel-in-time method to reduce the time-to-solution, as we now show.

3 | PARALLELIZATION OF THE STIELTJES PROCEDURE

3.1 | A brief introduction to PARAREAL

A classical sequential problem is obtained from the numerical solution of initial value problems of the form

$$\frac{d\boldsymbol{u}}{dt} = \mathcal{L}_h(\boldsymbol{u}(t), t), \quad \boldsymbol{u}(0) = \boldsymbol{u}_0, \quad t \in [0, T], \tag{20}$$

with $\mathcal{L}_h : \mathbb{R}^p \times \mathbb{R}^+ \to \mathbb{R}^p, \boldsymbol{u}(t) \in \mathbb{R}^p, \boldsymbol{u}_0 \in \mathbb{R}^p$, p being the total number of degrees of freedom and T a positive real number. Problem (20) often arises from the spatial discretization of a (non)linear system of partial differential equations (PDEs) through the method-of-lines. One then discretizes the problem in time by decomposing $[0,T]$ into several time steps, and computes an approximate solution for $\boldsymbol{u}$ one time step after the other. The solution at different time-steps can however also be computed in parallel through the more recent time parallelization approaches. We focus here on PARAREAL, and give a brief description of this algorithm; for more details and analogies, see Reference 10.

We decompose the global time interval $[0,T]$ into N time subintervals $[T_{n-1},T_n]$ of size ΔT, $n = 1,\dots,N$, where N is the number of processes to be considered for the time parallelization. In the following, we denote by $\boldsymbol{u}_n$ the approximation of $\boldsymbol{u}$ at time T_n, that is, $\boldsymbol{u}_n \approx \boldsymbol{u}(T_n)$. Let $\underset{T_{n-1}\to T_n}{\mathcal{F}^{\delta t}}(\boldsymbol{u}_{n-1})$ and $\underset{T_{n-1}\to T_n}{\mathcal{G}^{\Delta t}}(\boldsymbol{u}_{n-1})$ denote the action of approximately integrating (20) on the time subinterval $[T_{n-1},T_n]$ from a given starting value $\boldsymbol{u}_{n-1}$ using a fine propagator $\mathcal{F}$ with time step δt and a coarse propagator $\mathcal{G}$ with time step Δt, respectively. The PARAREAL algorithm requires the coarse propagator $\mathcal{G}$ to be much cheaper than the fine propagator, which induces a lower accuracy.

The PARAREAL algorithm consists of a prediction step and a correction iteration. In the prediction step, PARAREAL computes an initial guess of the starting values $\boldsymbol{u}_n^0$ at the beginning of each time subinterval using the coarse propagator,

$$\boldsymbol{u}_0^0 = \boldsymbol{u}_0, \quad \boldsymbol{u}_n^0 = \underset{T_{n-1}\to T_n}{\mathcal{G}^{\Delta t}}(\boldsymbol{u}_{n-1}^0), \quad \forall n = 1, \dots, N. \tag{21}$$

A correction iteration is then applied, using concurrently the fine propagator $\mathcal{F}$ on each time subinterval,

$$\boldsymbol{u}_n^k = \underset{T_{n-1}\to T_n}{\mathcal{F}^{\delta t}}(\boldsymbol{u}_{n-1}^{k-1}) + \underset{T_{n-1}\to T_n}{\mathcal{G}^{\Delta t}}(\boldsymbol{u}_{n-1}^{k}) - \underset{T_{n-1}\to T_n}{\mathcal{G}^{\Delta t}}(\boldsymbol{u}_{n-1}^{k-1}), \tag{22}$$

where $\boldsymbol{u}_n^k$ denotes the approximation of $\boldsymbol{u}$ at time T_n at the kth iteration of PARAREAL (k=1,…,K, $n = 1,\dots,N$).

The application of $\mathcal{F}$ can be performed independently for each time subinterval in parallel on different processors, which can speedup the computations using a parallel computer. But the expected speedup of PARAREAL remains limited by the sequential nature of the coarse integration performed by $\underset{T_{n-1}\to T_n}{\mathcal{G}^{\Delta t}}$ in (22), and how the algorithm itself is implemented. For PARAREAL, different implementations were thoroughly studied and tested in Reference 25. For illustration purposes, we give in Figure 3 a workflow diagram of an optimized implementation of PARAREAL for one specific configuration with N=6, where after a warm-up phase of doing the first coarse time step on each process, all $\mathcal{F}$ and $\mathcal{G}$ are then performed in parallel. We can see that if an accurate enough solution is obtained after K=3 iterations, then the computation using seven processors is faster than the sequential computation using only one processor (which only computes the solution for the four first time subintervals during the same time). PARAREAL will thus reduce the total computational time compared with a direct time-serial integration only if the application of $\mathcal{G}$ is cheap enough and if the total number of iterations K of PARAREAL is small compared with N. For the PARAREAL implementation represented in Figure 3, if the cost of $\mathcal{G}$ is negligible compared with $\mathcal{F}$, which we assume to have the same cost for each time subintervals, then the theoretical speedup of PARAREAL can be estimated[5] by

$$S(N,K) \simeq \frac{N}{K}. \tag{23}$$

It should be noted that approximation (23) only holds for very large problems, when communication time between processors is negligible compared with computation time for $\mathcal{G}$ and $\mathcal{F}$. Furthermore, it was assumed that the

[5]This is actually an upper bound on the parallel speedup of PARAREAL.

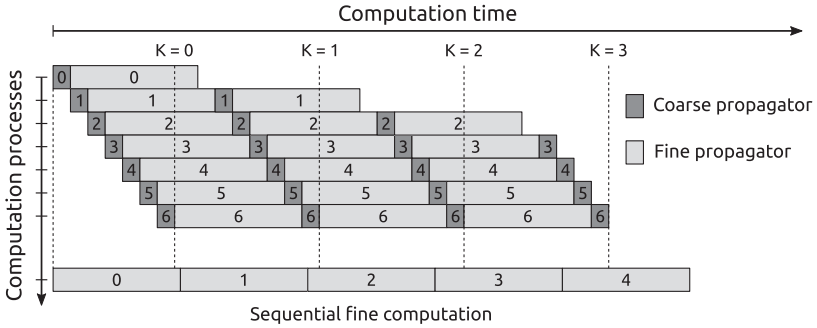

FIGURE 3 Workflow diagram of PARAREAL, for N=7, and comparison with a classical time-sequential computation. Number indicates the indexes of time subintervals, from 0 to $N-1$. Communication times between processes are not represented

computational cost of $\mathcal{G}$ is negligible. A comparison between theoretical and effective speedup of PARAREAL can be found, for example, in References 25,26.

A particularity of PARAREAL is that $\mathcal{F}$ and $\mathcal{G}$ are simple black box solvers that do only require a state vector $\boldsymbol{u}_{n-1}$ as entry, and give a similar state vector as output. The ODE properties are hidden in the propagator, and there is no influence of (20) in the initialization (21) and iteration (22) of PARAREAL. Hence, one can apply this algorithm to any sequential process for which one can define a decomposition into several sequential subintervals (equivalent to the $[T_{n-1},T_n]$) and a coarse and fine propagator for each subinterval.

3.2 | Parallel algorithm for the discrete Stieltjes procedure: PARASTIELTJES

For the Stieltjes procedure, we first consider the state vector at index j,

$$\boldsymbol{u}_j := [\alpha_j, \beta_j, \pi_j(t), \pi_{j-1}(t)]^T, \quad \text{with } 0 \leq j \leq J-1. \tag{24}$$

We then define as a first propagator the Stieltjes update, which uses a discrete quadrature rule with N_q points to compute the scalar products,

$$\mathcal{S}(\boldsymbol{u}_j) = \boldsymbol{u}_{j+1} = \begin{bmatrix} \alpha_{j+1} \\ \beta_{j+1} \\ \pi_{j+1}(t) \\ \pi_j(t) \end{bmatrix} = \begin{bmatrix} \langle t\pi_{j+1}, \pi_{j+1} \rangle_{N_q} / \langle \pi_{j+1}, \pi_{j+1} \rangle_{N_q} \\ \langle \pi_{j+1}, \pi_{j+1} \rangle_{N_q} / \langle \pi_j, \pi_j \rangle_{N_q} \\ (t-\alpha_j)\pi_j(t) - \beta_j \pi_{j-1}(t) \\ \pi_j(t) \end{bmatrix}, \tag{25}$$

and corresponds to one step of the discretized Stieltjes procedure described in Section 2.1. Furthermore, defining $\Delta_j \in \mathbb{N}$, we denote by

$$\mathcal{S}^{\Delta j}_{j \to j+\Delta j}(\boldsymbol{u}_j) := (\mathcal{S} \circ \mathcal{S} \circ \ldots \circ \mathcal{S})(\boldsymbol{u}_j) = \boldsymbol{u}_{j+\Delta j} \tag{26}$$

the successive applications of Δj Stieltjes updates to the state vector $\boldsymbol{u}_j$. We now can apply PARAREAL to the propagator $\mathcal{S}^{\Delta j}_{j \to j+\Delta j}$: each parallel subinterval is defined as an index interval $[j, j+\Delta j]$[6], and the fine and coarse propagators are defined similarly as $\mathcal{S}^{\Delta j}_{j \to j+\Delta j}$, using a discrete inner product with $N_{q,\mathcal{F}}$ and $N_{q,\mathcal{G}}$ points, such that $N_{q,\mathcal{F}} \gg N_{q,\mathcal{G}}$. This can be seen as equivalent to a coarsening in space for PARAREAL applied to a PDE, an idea that was originally proposed in Reference 27. We thus obtain a coarse and fine propagator that we can use for each PARAREAL iteration, as described in (22). The computation thus alternates between fine and coarse Stieltjes updates on the state vector at each parallel subinterval interface, in order to correct the initial approximation computed with the coarse Stieltjes procedure.

Finally, different discrete quadrature rules can be chosen for $\mathcal{F}$ and $\mathcal{G}$ (Fejer-I, asymptotic approximation, etc). We call this application of PARAREAL to the discrete Stieltjes procedure the PARASTIELTJES algorithm.

An important issue remains in the numerical representation of the polynomials π_j and π_{j-1} inside the state vector $\boldsymbol{u}_j$. We identified four possibilities to numerically represent the orthogonal polynomial π_j, but will focus here on the one that we consider best; a description of the other approaches is given in Appendix A1, along with a discussion of their main

[6]This interval is the equivalent of $[T_{n-1},T_n]$ in Section 3.1

advantages and drawbacks. Using (4), $\pi_j(t)$ can be computed at any $t \in [-1, 1]$ (in particular at the nodes of the discrete quadrature rule), as long as the recurrence coefficients are known up to the $(j-1)$th index. Hence knowing the first $(j-1)$th recurrence coefficients allows us to compute explicitly each scalar product in (25)[7]. So we define the vector representing our polynomials as

$$\boldsymbol{\pi}_j := [\alpha_0, \dots, \alpha_{j-1}, \beta_0, \dots, \beta_{j-1}]^T, \tag{27}$$

and the state vector in (24) becomes

$$\boldsymbol{u}_j = [\alpha_0, \dots, \alpha_j, \beta_0, \dots, \beta_j]. \tag{28}$$

3.3 | Computational cost and load balancing of ParaStieltjes

From the definition (28), we see that the size of the state vector $\boldsymbol{u}_j$ will grow with j, and thus also the computational cost of S. If we consider in ParaStieltjes a constant number of indices Δj for each subinterval, the propagator of the last subinterval will be substantially more costly computationally than the propagator on the first interval, and the theoretical speedup of ParaStieltjes would depend mostly on the computational cost of the last subinterval. A better approach is to distribute the ParaStieltjes updates (25) on the parallel subintervals, such that the computation time is equally distributed. This then optimizes the theoretical parallel speedup by reducing the idle time for each processor attributed to one subinterval. To do this, we need to predict the cost of one ParaStieltjes update for any index j, and for a first analysis, we chose to consider the cost to be dominated by floating point operations (FLOPs).

We denote by $\mathcal{C}$ the operator returning the cost, in FLOPS, of any computational element in (25). We have for any t in $[-1, 1]$

$$\mathcal{C}\left[\pi_j(t)\right] = 4(j-1) \quad \text{if } j > 1 \text{ else } 1. \tag{29}$$

Then, the cost of each discrete scalar product in (25) using N_q quadrature points is

$$\mathcal{C}\left[\left\langle t\pi_{j+1}, \pi_{j+1}\right\rangle_{N_q}\right] = N_q\left(\mathcal{C}\left[\pi_{j+1}(t)\right] + 3\right) + N_q - 1 = N_q(4j+4) - 1 \quad \text{if } j > 0 \text{ else } 5N_q - 1, \tag{30}$$

$$\mathcal{C}\left[\left\langle \pi_{j+1}, \pi_{j+1}\right\rangle_{N_q}\right] = N_q\left(\mathcal{C}\left[\pi_{j+1}(t)\right] + 2\right) + N_q - 1 = N_q(4j+3) - 1 \quad \text{if } j > 0 \text{ else } 4N_q - 1, \tag{31}$$

$$\mathcal{C}\left[\left\langle \pi_j, \pi_j\right\rangle_{N_q}\right] = N_q\left(\mathcal{C}\left[\pi_j(t)\right] + 2\right) + N_q - 1 = N_q(4j-1) - 1 \quad \text{if } j > 1 \text{ else } 4N_q - 1 \text{ if } j > 0 \text{ else } 0. \tag{32}$$

Finally adding to it the division performed to get the recurrence coefficients, we obtain for each given Stieltjes update (25) its cost in FLOPs

$$\mathcal{C}\left[S(\boldsymbol{u}_0)\right] = 9N_q, \tag{33}$$

$$\mathcal{C}\left[S(\boldsymbol{u}_1)\right] = 19N_q - 1, \tag{34}$$

$$\mathcal{C}\left[S(\boldsymbol{u}_j)\right] = 6N_q(2j+1) - 1, \quad \forall j > 1. \tag{35}$$

This gives us a first cost analysis of the ParaStieltjes propagators, in order to determine a load balanced partition that minimizes the computational cost difference between each subinterval. We will compare the convergence of ParaStieltjes with this load balanced partition to the one using a uniform number of updates on each subinterval in Section 4.2.

[7]Note that the evaluation of the orthogonal polynomials using the three-term recurrence formula (4) is generally numerically stable for any degree j.

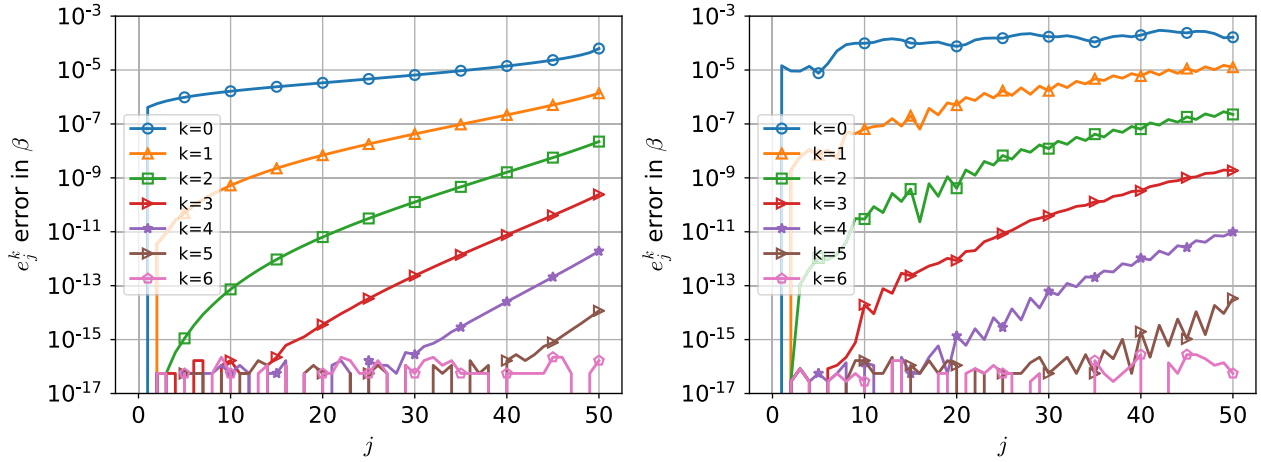

FIGURE 4 PARASTIELTJES absolute error compared with the fine sequential computation for the β recurrence coefficients, considering the JACOBI (left) and ELECTRO (right) weight functions

4 | NUMERICAL EXPERIMENTS

In this section, we investigate the convergence of the PARASTIELTJES algorithm developed in Section 3.2. For simplicity, we focus on the convergence of the algorithm for the β recurrence coefficients of (4), since the numerical results are similar for the α recurrence coefficients (some examples of convergence for the α recurrence coefficients are given in Appendix B1). We restrict our experiments to the computation of the 50 first recurrence coefficients ($J = 51$) of quadrature rules on $[-1, 1]$. For this number of coefficients, the Stieltjes procedure does not suffer from round-off errors and is numerically stable.

4.1 | First tests and proof of concept

We present now a first numerical convergence study of the PARASTIELTJES algorithm. We consider the application of one Stieltjes update on each parallel subinterval ($\Delta j = 1$, $N = 50$). We build our coarse propagator using a Fejer-I rule with $N_{q,\mathcal{G}} = 51$, $J = 102$, and our fine propagator using the asymptotic approximation of Section 2.3 with $N_{q,\mathcal{F}} = 50\,000$. We consider first the two weight functions JACOBI and ELECTRO, defined in (19) and (18). After each PARASTIELTJES iteration, we compute the absolute error at iteration k and subinterval index j, which gives for example for β

$$e_j^k = \left|\beta_j^{ref} - \beta_j^k\right|, \tag{36}$$

where β_j^{ref} represents the fine solution computed sequentially. This error is shown in Figure 4 for the first six PARASTIELTJES iterations ($K = 6$)[8]. We observe for both weight functions a rapid convergence of the algorithm over all coefficients. For the JACOBI weight function, PARASTIELTJES for $k = 4,5,6$ is down to machine precision for indices up to $j = 30,40,50$, respectively. For the ELECTRO weight function, we obtain a similar error reduction for $k = 5,6$. Even if the error of the coarse initialization ($k = 0$) differs for the two weight functions, it is rather similar when iterating, $k \geq 1$. Independently of the considered measure, only $K = 6$ iterations are needed to get the sequential error to machine precision for all β coefficients.

We next consider a more challenging example for the weight function, namely,

$$w(t) = (t < 0)\left(1 - \sqrt{|1 - (t+1)^2|}\right) + (t \geq 0)\left(1 - \sqrt{|1 - (t-1)^2|}\right), \tag{37}$$

which we call "FUNNEL" due to its particular shape shown in Figure 5 on the left. Even though the weight function is continuous on $[-1, 1]$, it has a sharp maximum with an infinite derivative at $t = 0$, which makes it hard for the coarse

[8]Since $\Delta j = 1$, the indices n of the parallel decomposition and the indices j of the recurrence coefficients coincide.

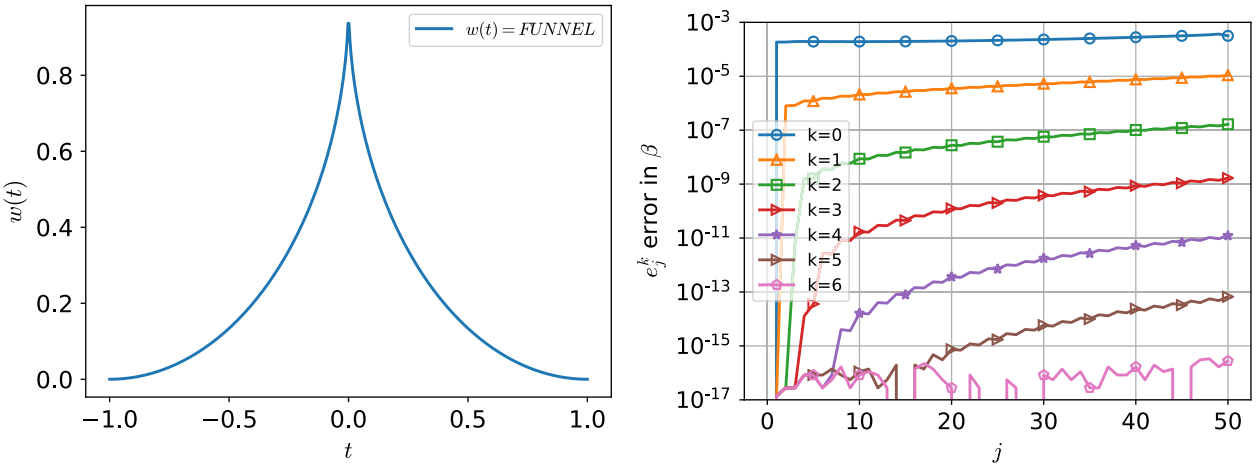

FIGURE 5 FUNNEL weight function (left), and corresponding PARASTIELTJES absolute error compared with the fine sequential computation for the β recurrence coefficient (right)

propagator to accurately compute the discrete scalar products. The error of the PARASTIELTJES algorithm compared with the fine sequential computation for the β coefficients are shown in Figure 5 on the right, with $K = 6$. We observe an important error for the coarse initialization ($k = 0$) that is almost constant in j. However, we still get rapid convergence for all the j indexes to machine precision after $k = 6$ iterations. This convergence behavior, which does not seem to depend on the weight function, is somehow an interesting property of the PARASTIELTJES algorithm, and was observed in other experiments with different weight functions, not shown here.

These first tests have shown a relative robustness of the PARASTIELTJES algorithm to compute the recurrence coefficients for nonclassical weight functions. However, for practical applications, several parameters of the PARASTIELTJES algorithm have to be set by the user, and we next study the influence of these parameters on the convergence of PARASTIELTJES.

4.2 | First analysis of the influence of PARASTIELTJES parameters

As explained in Section 3.2, several parameters have to be set by the user before applying PARASTIELTJES. Here we will focus on the size of the parallel subintervals Δj, and the number of quadrature points $N_{q,\mathcal{G}}$ used for the coarse discrete scalar product. For simplicity, we keep a Fejer-I rule for the coarse propagator, and the asymptotic approximation for the fine propagator with $N_{q,\mathcal{F}} = 50\ 000$. We chose to look at the maximum errors on all subinterval interface at each iteration k,

$$E_k = \max_{0 \le n \le N} \left| \beta_n^{\text{ref}} - \beta_n^k \right|, \tag{38}$$

where β_n^{ref} is the reference solution computed sequentially with the fine propagator. We use the JACOBI weight function in the following experiments. The error E_k for the results obtained in Section 4.1 for the JACOBI weight function is shown in Figure 6 on the left, with a dotted blue line with circle symbols. This is to be compared with the error curves for different coarse propagator accuracy (dotted lines of different color). Without surprise, the convergence of PARASTIELTJES improves when we increase $N_{q,\mathcal{G}}$ and thus the accuracy of the coarse solver. We also see that the error level increase is more important when $N_{q,\mathcal{G}}$ is divided by a factor 2 than the error level decrease when $N_{q,\mathcal{G}}$ is multiplied by a factor 2. The convergence rate is approximately linear, as the mean slopes indicate in Figure 6 on the left. We also see from these results that taking $N_{q,\mathcal{G}} = 2J$ already gives very good convergence for PARASTIELTJES, which divides the maximum error by roughly a factor 100 at each iteration then. This choice has proved to work well in all numerical experiments that we performed.

The other important parameter investigated in Figure 6 is the size of each parallel subinterval, given by the number of Stieltjes updates performed by one propagator (Δj). We use as reference again the error E_k for the results obtained in Section 4.1 for the JACOBI weight function shown in Figure 6 on the left with the dotted blue line with circle symbols, where we used $\Delta j = 1$ and $N = 50$. We compare this result to the convergence when using less parallel subintervals ($N = 10$), and two different distribution strategies: the first one, which we call "uniform," consists in keeping

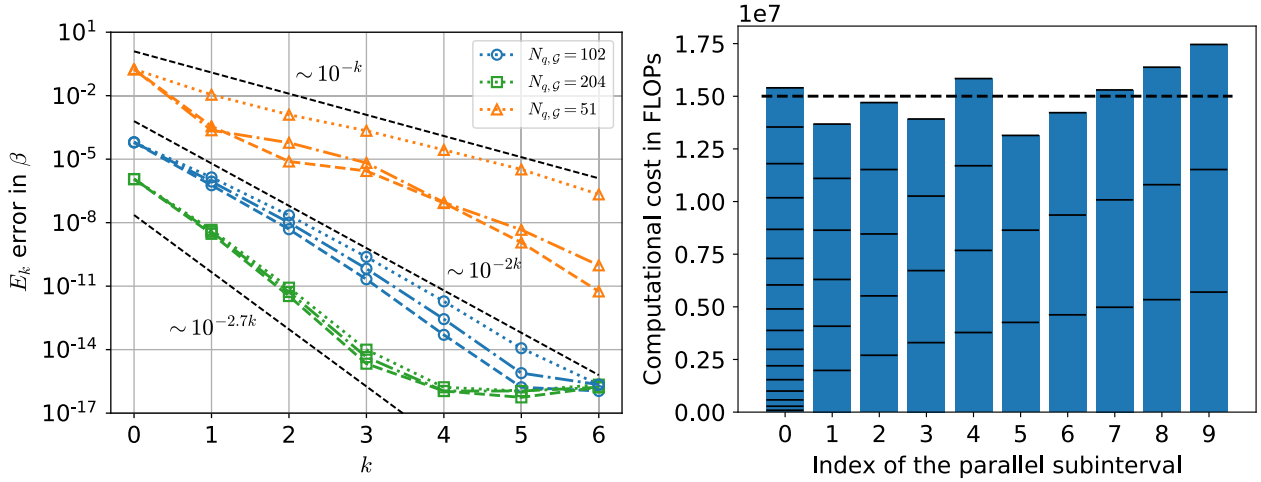

FIGURE 6 Influence of the coarse propagator accuracy and the subinterval decomposition on the convergence of PARASTIELTJES, for the JACOBI weight function. Left: error E_k for different values of $N_{q,\mathcal{G}}$ for the coarse propagator using a Fejer-I discrete rule, when the fine propagator uses the asymptotic approximation with $N_{q,\mathcal{F}} = 50\,000$, with a parallel decomposition with uniform numbers of indices on each subinterval, $N = 50$ and $\Delta j = 1$ (dotted lines), $N = 10$ and $\Delta_j = 5$ (dashed lines), and with a distribution inducing similar computational cost between subintervals (dash-dotted lines). Right: representation of the parallel decomposition with equivalent computational cost for all subintervals. Each bar represents the cost (in FLOPs) of each subinterval, with intermediate lines delimiting each Stieltjes update. The mean computational cost for each subinterval is indicated by the dashed line. The computational cost of the coarse propagators is neglected

Δj constant for each subinterval, that is, $\Delta j = 5$. As mentioned in Section 3.3, this approach causes the propagators on the last intervals to be the most computationally expensive, reducing the theoretical parallel speedup that can be obtained with PARASTIELTJES. Hence we also implemented a second distribution strategy, that tries to equally distribute the computational cost between parallel subintervals, with the help of the cost analysis given in Section 3.3. We obtain for this particular case the optimal distribution given by a Δj for each parallel subinterval of the size

$$\boldsymbol{\Delta j} = [16, 6, 5, 4, 4, 3, 3, 3, 3, 3]. \tag{39}$$

We show this decomposition in Figure 6 (right) by indicating the computational cost of each parallel subinterval and internal PARASTIELTJES update, considering only the cost of the fine propagator. This distribution is such that the computational cost difference between the slowest and fastest propagator (index 9 and 5 in Figure 6 on the right) is around 28.8% of the mean value for each parallel subinterval. We call this distribution strategy "optimal."

We compare the convergence of these two approaches with the reference experiments showing E_k[9] for each configuration in Figure 6 (left), while varying the value for $N_{q,\mathcal{G}}$. In all cases, the "uniform" (dashed lines) and "optimal" (dash-dotted lines) distribution strategies with $N = 10$ show a similar convergence behavior, better than the reference experiment with N=50. We also observe that the impact of the parallel distribution becomes negligible as soon as the coarse propagator is fine enough, and for $N_{q,\mathcal{G}} = 204$, the convergence curves are almost indistinguishable between the different parallel decompositions. This result may encourage the development of a convergence theory that will only consider uniform distributions, which could give relatively accurate convergence bounds also for the nonuniform distributions.

4.3 | Further analysis: Link between error and residual

The error of PARASTIELTJES with the fine sequential solution is not known during the computation, and can thus not be used as a stopping criterion for PARASTIELTJES. One can only look at criteria obtained during the PARASTIELTJES iterations, one of which is the residual at iteration k and parallel subinterval index n (eg, for β),

$$r_n^k = \left|\beta_n^k - \beta_n^{k-1}\right|. \tag{40}$$

[9]We show further results for the absolute error e_j^k in the β coefficients for each configuration in Appendix C1, to be compared with the results given in Figure 4.

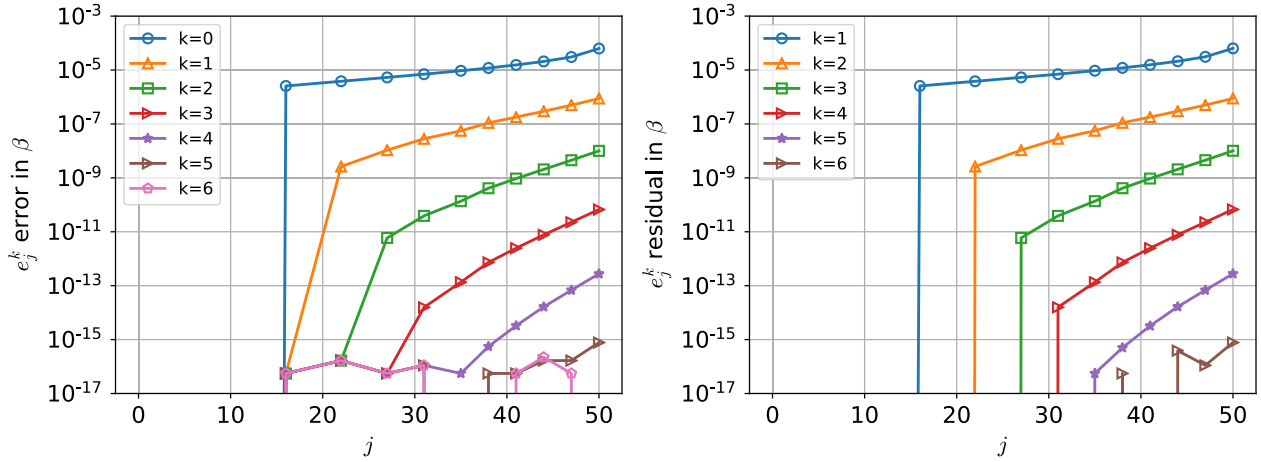

FIGURE 7 Similarity between the sequential error and residual using PARASTIELTJES for the JACOBI weight function, with the optimal distribution with $N = 10$, $\mathcal{G}$ using $N_{q,\mathcal{G}} = 102$ and a Fejer-I, $\mathcal{F}$ using $N_{q,\mathcal{F}} = 50\ 000$ and the asymptotic approximation. Plot of the sequential error (left) and the residual (right) for the six first iterations of PARASTIELTJES. Each curve symbol represent the error at the interface of two parallel subintervals

Naturally, one must perform at least one iteration of PARASTIELTJES to compute this residual. We show in Figure 7 the sequential error e_n^k (36) along with the residual r_n^k for the first six iterations for the JACOBI weight function, with the coarse propagator using $N_{q,\mathcal{G}} = 102$ and Fejer-I, and the fine propagator using $N_{q,\mathcal{F}} = 50\ 000$ and the asymptotic approximation.

We see that e_n^k and r_n^{k+1} are very similar, and this for all subinterval interfaces. We observed this similarity for all experiments we performed. Hence, looking at the residual gives an accurate approximation of the sequential error of the previous iteration. Combining this with some linear bound on the error (cf Section 4.2) would then give a reasonable estimate of the current sequential error, and allow the user to stop the iteration process when a given error criterion is reached.

5 | FURTHER APPLICATIONS AND DISCUSSION

5.1 | Highly accurate on-the-fly computation of orthogonal polynomials

The PARASTIELTJES algorithm can be used to compute orthogonal polynomials, associated with a weight function that varies during the computation. For example, this occurs when solving partial differential equations with stochastic inputs using the multielement generalized polynomial chaos (ME-gPC) method,[28] which models random processes by projecting them onto a basis of orthogonal polynomials (the so-called polynomial chaos). During the computation, random vector variables go through an update process where they are projected onto a renewed orthogonal basis associated with a changing weight function. For this purpose, the authors in Reference 28, sec. 2.2.2 use the discetization method of Gautschi and the Stieltjes procedure[28(sec. 2.2.2)], as described in Section 2.1. Although the computation of the orthogonal basis does not represent the main part of the computation time for ME-gPC, the accuracy of the Steiltjes procedure is important[29(sec. 2.7)]. The use of PARASTIELTJES in this context could provide high accuracy while still keeping a low time-to-solution.

There are many applications based on orthogonal polynomials that can benefit from the use of nonclassical measures (eg, interpolation, approximation and quadrature,[30] spectral methods,[31] Lagrange mesh[32]). Those could be extended to the use of singular or discontinuous weight functions, but the use of the Stieltjes procedure combined with the discretization method would make the computation of the associated orthogonal polynomials particularly expensive if high accuracy is needed. The use of PARASTIELTJES in a parallel computing environment could be a remedy.

5.2 | Discussion on PARASTIELTJES speedup

From the numerical experiments in Section 4, we consider the configuration using $N = 10$, the optimized distribution for parallel subintervals with Δ_j shown in (39), and $N_{q,\mathcal{G}} = 204 \ll N_{q,\mathcal{F}}$. This corresponds to the convergence results given

in Figure 6 (left). The computational cost of $\mathcal{G}$ is negligible compared with $\mathcal{F}$, and $\mathcal{F}$ has about the same cost over all parallel subintervals. For a first theoretical speedup estimate, we assume that communication time between processes is negligible, so we can use (23). If we stop after $K = 3$ iterations with an error around 1e−14, this gives a theoretical speedup $S(N = 10, K = 3) \simeq 3.3$, comparable with theoretical speedups of PARAREAL. In practice, concrete speedup results are however influenced by further important factors:

1. overhead due to communication between processors (making practical speedup worse),
2. a small but certain computational workload due to the coarse solver $\mathcal{G}$ (making practical speedup worse),
3. an optimization of the ratio $N_{q,\mathcal{G}}/N_{q,\mathcal{F}}$ and the minimum number of iterations K to achieve a targeted accuracy (making practical speedup better).

Only concrete parallel speedup measurements in an HPC setting with an optimized implementation can fully do justice to answer the question of the speedup potential of PARASTIELTJES; for PARAREAL, such results can be found in References 25,26.

6 | CONCLUSION

We presented a new algorithm for the computation of Gauss quadrature rules with nonclassical weight functions. This approach, which we call the PARASTIELTJES algorithm, combines the Stieltjes procedure and the discretization method developed by Gautschi,[19] and parallelizes it through the PARAREAL algorithm.[9] A complete description of PARASTIELTJES is given in Section 3, and numerical experiments were performed in Section 4. These showed a good robustness of the algorithm for different weight functions, and the influence of the main parameters on its convergence. Finally, we discussed further potential applications and implementation considerations in Section 5.

PARASTIELTJES is a promising algorithm for the rapid and parallel computation of general Gauss quadrature rules to very high accuracy, which benefits from the widely available parallel computing resources and newly developed parallel solution techniques. PARASTIELTJES also shows that time-parallelization algorithms are not restricted to the solution of time-dependent problems: any expensive computation process that is inherently sequential may benefit from a time-parallel technique, and other methods than PARAREAL could be considered. For instance, the MGRIT algorithm, which is equivalent to PARAREAL in some settings,[10] can also be considered in order to extend the parallel computation with a multilevel approach. Indeed MGRIT has been considered to train neural networks in Reference 8, which are also not really representing time-dependent problems. The scientific community is currently still developing the understanding and efficiency of such time-parallel algorithms, and most of the existing solution techniques and knowledge on these algorithms are accessible through a community website dedicated to this topic.[33]

We clearly also have future work to do for PARASTIELTJES: as mentioned in Section 4.2, a theoretical convergence analysis is planned in order to accompany future applications. Different approaches for the coarse propagator of PARASTIELTJES should be investigated, such as adding a coarsening through the step of the sequential process. Also, the application of PARASTIELTJES for quadrature rules on unbounded intervals is of great interest. Prediction and measurement of the effective parallel speedup of an efficient implementation of PARASTIELTJES are also future work, based on the existing developments for PARAREAL.[25,26] Finally, the development of a general-purpose library that would make the algorithm available to the community is also considered.

ACKNOWLEDGEMENTS

The authors thank two anonymous reviewers for the time they took for making helpful comments that allowed us to improve the quality of this manuscript. Finally, this work does not have any conflicts of interest.

ORCID

Thibaut Lunet https://orcid.org/0000-0003-1745-0780

REFERENCES

1. Den Iseger P. Numerical transform inversion using Gaussian quadrature. Probab Eng Inf Sci. 2006;20(1):1–44.
2. Gander MJ, Karp AH. Stable computation of high order Gauss quadrature rules using discretization for measures in radiation transfer. J Quant Spectrosc Radiat Transf. 2001;68(2):213–223.

3. Gautschi W. Algorithm 726: ORTHPOL–a package of routines for generating orthogonal polynomials and Gauss-type quadrature rules. ACM Trans Math Softw (TOMS). 1994;20(1):21–62.
4. Gander MJ. 50 years of time parallel time integration. In: Carraro T, Geiger M, Körkel S, Rannacher R, editors. Multiple shooting and time domain decomposition methods. New York, NY: Springer, 2015; p. 69–114.
5. Lecouvez M, Falgout RD, Woodward CS, Top P. A parallel multigrid reduction in time method for power systems. Proceedings of the 2016 IEEE Power and Energy Society General Meeting (PESGM); 2016 p. 1–5.
6. Götschel S, Minion ML. Parallel-in-time for parabolic optimal control problems using PFASST. New York, NY: Springer, 2017; p. 363–371.
7. Lunet T, Bodart J, Gratton S, Vasseur X. Time-parallel simulation of the decay of homogeneous turbulence using Parareal with spatial coarsening. Comput Vis Sci. 2018;19(1-2):31–44.
8. Schroder JB. Parallelizing over artificial neural network training runs with multigrid; 2017. arXiv preprint arXiv:170802276.
9. Lions JL, Maday Y, Turinici G. A "Parareal" in time discretization of PDE's. C R Math Acad Sci Paris. 2001;332(7):661–668.
10. Gander MJ, Vandewalle S. Analysis of the Parareal time-parallel time-integration method. SIAM J Sci Comput. 2007;29(2): 556–578.
11. Gander MJ, Hairer E. Nonlinear convergence analysis for the parareal algorithm. In: Widlund OB, Keyes DE, editors. Domain decomposition methods in science and engineering XVII. vol. 60 of lecture notes in computational science and engineering. New York, NY: Springer, 2008; p. 45–56.
12. Gander MJ. Analysis of the Parareal algorithm applied to hyperbolic problems using characteristics. Bol Soc Esp Mat Apl. 2008;42: 21–35.
13. Gautschi W. Orthogonal polynomials: computation and approximation. Oxford, UK: Oxford University Press, 2004.
14. Golub GH, Welsch JH. Calculation of Gauss quadrature rules. Math Comput. 1969;23(106):221–230.
15. Gander W, Gander MJ, Kwok F. Scientific computing-an introduction using maple and MATLAB. Vol 11. Luxemburg: Springer Science & Business, 2014.
16. Townsend A. The race for high order Gauss–Legendre quadrature. SIAM News. 2015;48:1–3.
17. Golub GH, Meurant G. Matrices, moments and quadrature with applications. Princeton, NJ: Princeton University Press, 2009.
18. Gautschi W. On generating orthogonal polynomials. SIAM J Sci Stat Comput. 1982;3(3):289–317.
19. Gautschi W. Construction of Gauss-Christoffel quadrature formulas. Math Comput. 1968;22(102):251–270.
20. Gautschi W. Computational problems and applications of orthogonal polynomials. Orth Polynom Appl. 1991;9:61–71.
21. Gautschi W. The circle theorem and related theorems for Gauss-type quadrature rules. Electron Trans Numer Anal. 2006;25: 129–137.
22. Nevai P. Géza Freud, orthogonal polynomials and Christoffel functions. a case study. J Approx Theory. 1986;48(1):3–167.
23. Akhiezer N. On a theorem of academician SN Bernstein concerning a quadrature formula of PL Chebyshev (Ukrainian), Zh. Inst Mat Akad Nauk Ukrain RSR. 1937;3:75–82.
24. Szegö G. Orthogonal Polynomials. Colloquium Publications, Providence, Rhode Island: American Mathematical Soc; 1939. 23.
25. Aubanel E. Scheduling of tasks in the Parareal algorithm. Parallel Comput. 2011;37(3):172–182.
26. Ruprecht D. Shared memory pipelined Parareal. Paper presented at: Proceedings of the European Conference on Parallel Processing. Springer; 2017. p. 669–681.
27. Fischer PF, Hecht F, and Maday Y. A Parareal in time semi-implicit approximation of the Navier-Stokes equations. In: Kornhuber R ed. Domain decomposition methods in science and engineering, lecture notes in computational science and engineering. vol. 40. New York, NY: Springer; 2005. p. 433–440.
28. Wan X, Karniadakis GE. Multi-element generalized polynomial chaos for arbitrary probability measures. SIAM J Sci Comput. 2006;28(3):901–928.
29. Zheng M, Wan X, Karniadakis GE. Adaptive multi-element polynomial chaos with discrete measure: Algorithms and application to SPDEs. Appl Numer Math. 2015;90:91–110.
30. Gautschi W. Orthogonal polynomials: Applications and computation. Acta Numer. 1996;5:45–119.
31. Weideman JAC. Spectral methods based on nonclassical orthogonal polynomials. Applications and computation of orthogonal polynomials. New York, NY: Springer, 1999; p. 239–251.
32. Baye D, Vincke M. Lagrange meshes from nonclassical orthogonal polynomials. Phys Rev E. 1999;59(6):7195.
33. Parallel-in-time community website. http://parallel-in-time.org/. Accessed May 06, 2019.

How to cite this article: Gander MJ, Lunet T. PARASTIELTJES: Parallel computation of Gauss quadrature rules using a PARAREAL-like approach for the Stieltjes procedure. *Numer Linear Algebra Appl*. 2021;28:e2314. https://doi.org/10.1002/nla.2314

APPENDIX A. NUMERICAL REPRESENTATION OF ORTHOGONAL POLYNOMIALS

We give here three alternative numerical representations for the orthogonal polynomials that can be used for the definition of the Stieltjes update in (25).

The first approach consists in representing the polynomials at each quadrature point of the discrete scalar product (9), that is,

$$\boldsymbol{\pi}_j := [\pi_j(\tau_1^D), \dots, \pi_j(\tau_{N_q}^D)]. \tag{A1}$$

This formulation has the advantage of keeping a constant cost of the Stieltjes update with j; however in the context of using a coarse and a fine propagator with $N_{q,\mathcal{G}} \neq N_{q,\mathcal{F}}$, this requires the use of transfer operators (interpolation and restriction) between the fine and the coarse quadrature points for the PARAREAL update (22). Those transfer operators induce an increase in the computational cost, but also a notable loss in the accuracy for values of j that are bigger than the interpolation order.

The second approach avoids the use of transfer operators by representing a polynomial by its coefficients, that is,

$$\pi_j(t) = c_0 + c_1 t + \dots + c_{j-1} t^{j-1} + t^j \quad \Rightarrow \quad \boldsymbol{\pi}_j := [c_0, \dots, c_{j-1}]. \tag{A2}$$

Then the computation of the Stieltjes update at index j requires to evaluate this polynomial at the discrete quadrature nodes to compute the scalar products, and uses an explicit formula to get the polynomial coefficients of π_{j+1} from (4). However, numerical instabilities arise when numerically evaluating the polynomial with the Horner scheme for large values of j, as the oscillatory behavior of the orthogonal polynomials becomes more pronounced. This limits the application of PARASTIELTJES to a given maximum value of j, around 15 as we observed in our experiments.

The third approach consists in representing the polynomial by its Newton form, that is,

$$\pi_j(t) = (t - \tau_1) \dots (t - \tau_J) \quad \Rightarrow \quad \boldsymbol{\pi}_j := [\tau_1, \dots, \tau_J]. \tag{A3}$$

This has the advantage of already computing the roots of each orthogonal polynomials during the Stieltjes procedure, and it does not need any transfer operators between the fine and coarse propagator. Furthermore, evaluating this polynomial at a given t is less computationally expensive than the formulation using the three-term recurrence formula. However, the main drawback arises when computing the Stieltjes update: at index j, one needs to solve a system of $j+1$ nonlinear equations to find all the roots of π_{j+1} using (4), increasing substantially the computational cost of the Stieltjes update. One can use for instance Newton's method for this, combined with the asymptotic approximation for the initial guess of the roots. But the PARASTIELTJES algorithm using this polynomial formulation was in our experiments still the most expensive approach, and even numerically unstable for some particular weight functions.

APPENDIX B. CONVERGENCE OF PARASTIELTJES FOR THE α RECURRENCE COEFFICIENTS

We now illustrate the convergence of the PARASTIELTJES algorithm for the α recurrence coefficients of (4). These results are to be compared with those given in Figure 4, as the same experimental setting described in Section 4.1 is used, and also the same scales for both Figures 4 and B1, where we plot the absolute error (36) for the recurrence coefficients α. The main difference we observe is for the sequential error of the initialization ($k = 0$), for the other iterations $k \geq 1$ the errors are similar to the ones obtained for the β recurrence coefficients, especially if we consider only the maximum error E_k over all indices j. Similar results were obtained with numerical experiments also for the other weight functions.

APPENDIX C. CONVERGENCE OF PARASTIELTJES FOR THE β RECURRENCE COEFFICIENTS WITH VARIOUS PARALLEL SUBINTERVAL DISTRIBUTIONS

We give here the absolute error e_j^k of PARASTIELTJES for the β coefficients, considering the JACOBI and the ELECTRO weight functions, using the same configuration as described in Section 4.2. For $N = 10$, we show in Figure C1 the results for a uniform distribution of the parallel subintervals ($\Delta_j = 5$), and in Figure C2 for an optimized distribution with variable Δ_j defined in (39). The results for the JACOBI weight function can be compared with those for the β coefficients given in Figure 6.

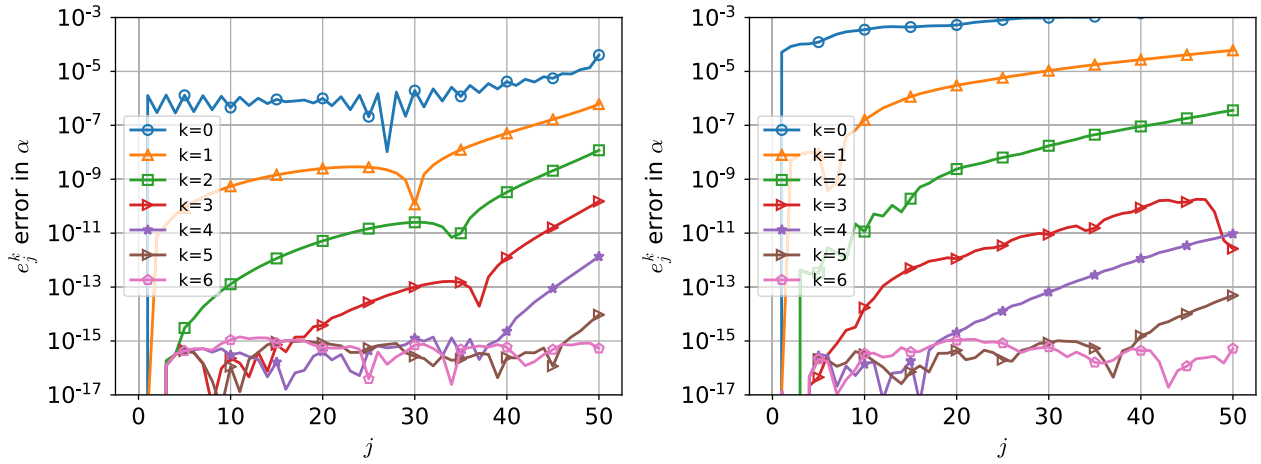

FIGURE B1 PARASTIELTJES error compared with the fine sequential computation for the α recurrence coefficients, considering the JACOBI (left) and ELECTRO (right) weight functions

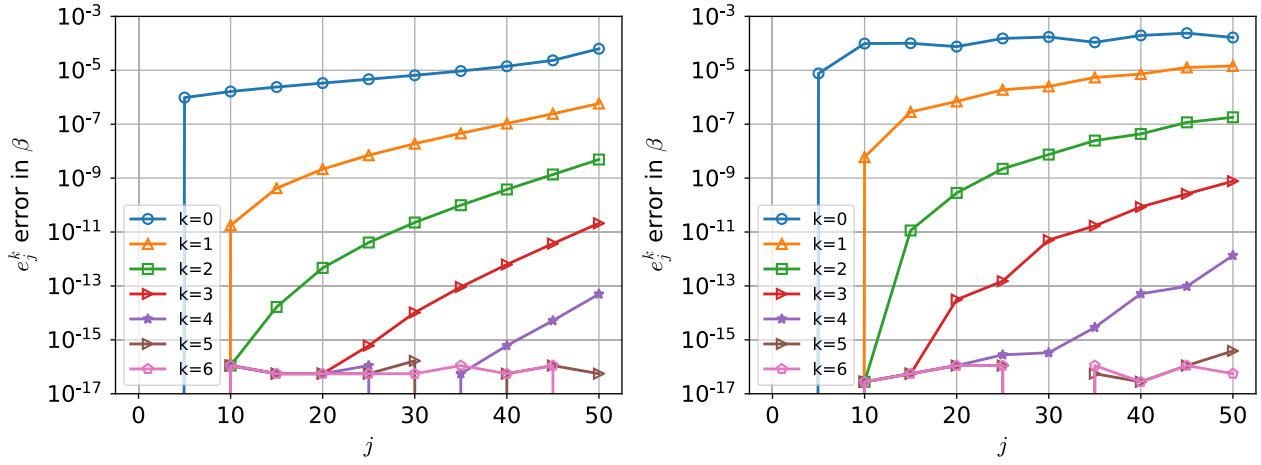

FIGURE C1 PARASTIELTJES error compared with the fine sequential computation for the β recurrence coefficients, considering the JACOBI (left) and ELECTRO (right) weight functions, and a uniform distribution for the parallel subintervals ($N = 10$, $\Delta_j = 5$). Each curve symbol represents the error at the interface of two parallel subintervals

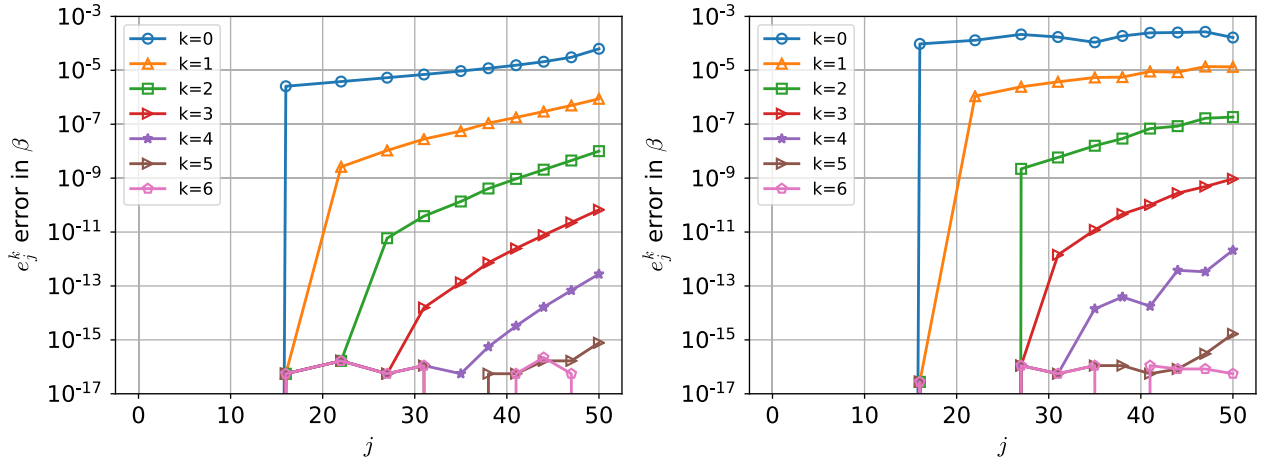

FIGURE C2 PARASTIELTJES error compared with the fine sequential computation for the β recurrence coefficients, considering the JACOBI (left) and ELECTRO (right) weight functions, and the optimal distribution for the parallel subintervals (Δ_j given in (39)). Each curve symbol represents the error at the interface of two parallel subintervals