ORTHOPOLY: A library for accurate evaluation of series of classical orthogonal polynomials and their derivatives

Roberto Barrio[a,*], Peibing Du[b], Hao Jiang[c], Sergio Serrano[a]

[a]*Departamento de Matemática Aplicada and IUMA. University of Zaragoza, E-50009 Zaragoza, Spain.*
[b]*School of Science, National University of Defense Technology, Changsha, 410073, China.*
[c]*School of Computer, National University of Defense Technology, Changsha, 410073, China.*

Abstract

We present the ORTHOPOLY software that permits to evaluate, efficiently and accurately, finite series of any classical family of orthogonal polynomials (Chebyshev, Legendre, ultraspherical or Gegenbauer, Jacobi, Hermite and Laguerre orthogonal polynomials) and their derivatives. The basic algorithm is the *BCS-algorithm* (Barrio-Clenshaw-Smith derivative algorithm), that permits to evaluate the k-th derivative of a finite series of orthogonal polynomials at any point without obtaining before the previous derivatives. Due to the presence of rounding errors, specially in the case of high order derivatives, we introduce the *compensated BCS-algorithm*, based on Error-Free Transformation techniques, that permits to relegate the influence of the conditioning of the problem up to second order in the round-off unit of the computer. The BCS and compensated BCS algorithms may also give running-error bounds to provide information about the accuracy of the evaluation process. The ORTHOPOLY software includes C and `Matlab` versions of all the algorithms, and they are designed to be easily used in longer softwares to solve physical, mathematical, chemical or engineering problems (illustrated on the Schrödinger equation for the radial hydrogen atom).

Keywords: Classical orthogonal polynomials, evaluation algorithms, derivative evaluation, accurate algorithms, Error-Free Transformation techniques

PROGRAM SUMMARY

Program Title: ORTHOPOLY
Licensing provisions(please choose one): GPLv3
Programming language: C and `Matlab` versions
Nature of problem(approx. 50-250 words): Accurate numerical evaluation of finite series of classical orthogonal polynomials and their derivatives.
Solution method(approx. 50-250 words): Barrio-Clenshaw–Smith algorithm for the evaluation of derivatives of finite series of classical orthogonal polynomials. Error-Free Transformation techniques for the *Compensated Barrio-Clenshaw–Smith algorithm* in order to provide accurate evaluations. Running-error techniques to provide error bounds of the evaluations.

1. Introduction

The different families of classical orthogonal polynomials (COPs) are nowadays part of the basic mathematical machinery of numerous engineering, physical and mathematical algorithms and methodologies. For example (see [1] for more details), the Toda equation provides an important model of a completely integrable system and by using Hirota's technique of bilinear formalism of soliton theory, Nakamura [2] shows that a wide class of exact solutions of the Toda equation can be expressed in terms of classical orthogonal polynomials. Hermite, Legendre and Laguerre orthogonal polynomials appear when the time-dependent Schrödinger equation is solved by separation of variables [3, 4, 5]. In potential theory, Gegenbauer polynomials, as they are zonal spherical harmonics, have many applications.

[*]Corresponding author.
E-mail address: rbarrio@unizar.es

Hermite polynomials are known to play an important role in random matrix theory. Once the different families of COPs are used, it is typical to use finite series of such polynomials. Finite series of COPs appear, for example, in the approximation of functions, in the integration of ODEs and PDEs by means of the collocation method, in nuclear physics, etc.

Also, the need to evaluate the derivatives of a polynomial at a specific point arises in many contexts, a typical example being that of finding the roots of polynomial equations. Several algorithms have been elaborated for the derivative evaluations, and the most obvious consists on determining the derivative of the polynomial and afterwards evaluating it at the point x. A classical way to evaluate orthogonal polynomial series is the Clenshaw-Smith algorithm [6, 7]. This algorithm provides an alternative way to compute the derivatives proposed by Smith [7], that needs also to evaluate all the precedent derivatives, obtained just by derivation of the recurrence, as in [8, 9]. Other options to evaluate derivatives of an orthogonal polynomial series come from the pseudospectral methods for the solution of PDEs [4, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20], where it is necessary to evaluate such derivatives. The corresponding derivatives are usually computed by means of the so-called differentiation matrices. For example, in classical collocation methods it is necessary to evaluate the derivative of Chebyshev or Legendre series in special sets of points. For that situation, explicit expressions of the first and second order differentiation matrices have been given [10, 11, 15, 16, 21]. Besides, other methods to use derivatives in collocation methods have been proposed more recently, like the method of Baltensperger and Berrut based on the rational interpolation method of Schneider and Werner [13] and the precomputed variation of the method used by Tang and Trummer [22]. Finally, we remark the recent interest in using Laguerre and Hermite families of orthogonal polynomials in several fields [4, 19, 20], and the necessity of new algorithms to deal with them [23].

In this paper we present the ORTHOPOLY software to evaluate, efficiently and accurately, finite series of any classical family of orthogonal polynomials (Chebyshev polynomials of the first and second kind, Legendre, ultraspherical or Gegenbauer, Jacobi, Hermite and generalized Laguerre orthogonal polynomials) and their derivatives. The basic algorithm is the BCS-algorithm (Barrio-Clenshaw-Smith derivative algorithm [24, 25]), that permits to evaluate the k-th derivative ($k \geq 0$) of a finite series of COPs at any point $x \in D$ without obtaining before the previous derivatives, and independently of the number and type of points of evaluation. This algorithm presents rounding error bounds similar to the best methods currently used in literature [25]. Due to the presence of rounding errors, specially in the case of high order derivatives, we also introduce a modification of the BCS-algorithm, the compensated BCS-algorithm, based on Error-Free Transformation techniques [26], that relegates the influence of the conditioning of the problem up to second order in the round-off unit of the computer. This fact permits to evaluate accurately the derivatives of the polynomials in most cases. This algorithm is especially useful when we have to evaluate derivatives of high order. This happens, for example, in spectral methods used to solve fourth order PDEs [27, 28, 29] such as the Navier-Stokes equation with a viscosity term, the Kuramoto-Sivashinsky equation, etc. For these problems the rounding error can ruin the numerical solution, even for small numbers of collocation points. Moreover, in recent literature more and more care is paid to high precision simulations [30]. All these algorithms are implemented in the ORTHOPOLY software, also with running-error bounds that give to the user some information about the quality of the evaluation results.

The paper is organized as follows; in Section 2 we first review some basic properties of the classical orthogonal polynomials and the Barrio-Clenshaw–Smith (BCS) algorithm [7, 24, 25], and its running-error bound, which permits to evaluate, in an efficient way, series of orthogonal polynomials and their derivatives. Next, in Section 3 we propose a highly accurate version of the BCS-algorithm, the compensated BCS-algorithm. In Section 4 we describe the ORTHOPOLY software structure and its basic use. Several numerical tests are shown in Section 5 using Chebyshev, Gegenbauer, Jacobi and generalized Laguerre orthogonal polynomial series. In Section 6 a simple example of the use of the ORTHOPOLY library on the Schrödinger equation for the radial hydrogen atom illustrates its use in a physical problem. Finally, in the Appendices, two example files, in C and `Matlab`, are shown.

2. Theoretical background

2.1. *Classical orthogonal polynomials*

In this paper we consider the classical families of orthogonal polynomials: Chebyshev polynomials of the first ($T_n(x)$) and second kind ($U_n(x)$), Legendre ($P_n(x)$), ultraspherical or Gegenbauer (C_n^{λ}, with $\lambda > -1/2$), Jacobi ($P_n^{(\alpha,\beta)}$, with $\alpha,\beta > -1$), Hermite (H_n and He_n) and generalized Laguerre ($L_n^{(\alpha)}$, with $\alpha > -1$) orthogonal polynomials. These families are very important in theoretical and practical applications, as indicated in the introduction. All these families are characterized by their orthogonality condition [1] with respect to some weight functions in different intervals D ($D \equiv [-1, 1]$ for the Chebyshev, Legendre, Gegenbauer and Jacobi polynomials, $D \equiv (0, +\infty)$ for the generalized Laguerre polynomials, and $D \equiv \mathbb{R}$ for the Hermite polynomials). Moreover, they are solutions of some second order differential equations and all of them satisfy a three-term recurrence relation:

$$\begin{aligned} &p_0 = 1, \qquad p_1 = A_1^{1,0}(x), \\ &p_j(x) - A_j^{1,0}(x)\, p_{j-1}(x) - A_j^{2,0}\, p_{j-2}(x) = 0, \quad j = 2, \ldots, n, \end{aligned} \tag{1}$$

with $A_j^{1,0}(x)$ a first degree polynomial and $A_j^{2,0}$ a constant (in Tables 1 and 2 a complete list of the terms $A_j^{1,k}(x)$ and $A_j^{2,k}$ are also given just considering $k = 0$).

In particular, the recurrences for the Chebyshev (first kind), Gegenbauer and generalized Laguerre polynomials are given by

$$\begin{aligned} T_j(x) &= 2x\,T_{j-1}(x) - T_{j-2}(x), \quad T_0(x) = 1, \quad T_1(x) = x, \\ C_j^{\lambda}(x) &= \frac{2(j+\lambda-1)}{j}\,x\,C_{j-1}^{\lambda}(x) - \frac{j+2\lambda-2}{j}\,C_{j-2}^{\lambda}(x), \quad C_0^{\lambda}(x) = 1, \quad C_1^{\lambda}(x) = 2\lambda x, \\ L_j^{\alpha}(x) &= \left(-\frac{1}{j+1}\,x + \frac{2j+\alpha+1}{j+1}\right) L_{j-1}^{\alpha}(x) - \frac{j+\alpha}{j+1} L_{j-2}^{\alpha}(x), \quad L_0^{\alpha}(x) = 1, \quad L_1^{\alpha}(x) = 1 + \alpha - x. \end{aligned}$$

Another important property for our purposes, that gives us the expression of the derivative of the polynomials, is (see [1] for a complete list of derivative relations)

$$\begin{aligned} \frac{d}{dx} T_j(x) &= j\,U_{j-1}(x), \\ \frac{d^k}{dx^k} C_j^{\lambda}(x) &= 2^k\,(\lambda)_k\,C_{j-k}^{\lambda+k}(x), \qquad \text{for} \quad \lambda \neq 0, \\ \frac{d^k}{dx^k} L_j^{\alpha}(x) &= (-1)^k L_{j-k}^{\alpha+k}(x), \end{aligned} \tag{2}$$

where $(z)_m = z \cdot (z+1) \cdots (z+m-1)$ ($(z)_0 \doteq 1$) is the Pochhammer symbol.

The evaluation of a finite series of any of these polynomials $p(x) = \sum_{j=0}^{n} c_j\, p_j(x)$ at any x can be done by means of the Smith generalization [7] of the Clenshaw's algorithm [6].

2.2. Evaluation algorithms

In [24, 25], using the derivative properties of the classical orthogonal polynomials and the Clenshaw-Smith algorithm, it was proposed an extended algorithm (the Barrio-Clenshaw-Smith (BCS) algorithm) that allows to evaluate finite polynomial series, but also any derivative of them. The algorithm is based on the relations of the derivatives of the different families of classical orthogonal polynomials (2). For instance, the Chebyshev polynomials of the first kind are related to the Gegenbauer polynomials by means of the relation $T_j(x) = \frac{j}{2}\, C_j^0(x)$, thus, differentiating [1]

$$\frac{d^k}{dx^k}\, T_j(x) = 2^{k-1}\,\Gamma(k)\, j\, C_{j-k}^{k}(x) = 2^{k-1}\,(k-1)!\, j\, C_{j-k}^{k}(x).$$

Therefore, the evaluation of the k-th derivative of a finite series of Chebyshev polynomials is equivalent to the evaluation of a finite series of Gegenbauer polynomials with different coefficients. Similar relations are satisfied by all the classical orthogonal polynomials [1]. This process gives rise to the BCS-algorithm [24, 25] for evaluating derivatives of orthogonal polynomial series: let $p(x) = \sum_{j=0}^{n} c_j\, p_j(x)$, where $\{p_j(x)\}$ is an orthogonal polynomial basis satisfying (1), a point $y \in D \subseteq \mathbb{R}$, and $k \in \mathbb{N} \cup \{0\}$ the derivation order, then, using coefficients C^k, $A_j^{c,k}$, $A_j^{1,k}$ and $A_j^{2,k}$ for the different families of classical orthogonal polynomials given in Tables 1 and 2 we have

Algorithm 1. *BCS-algorithm [24, 25] to evaluate the k-th derivative of an orthogonal polynomial series*

```
function BCS$(p, y, k)$
  $q_{n-k+2} = q_{n-k+1} = 0$
  for $j = n - k : -1 : 0$
      $q_j = A_j^{c,k}\, c_{j+k} + A_j^{1,k}(y)\, q_{j+1} + A_j^{2,k}\, q_{j+2}$
  end
  $\mathrm{BCS}(p, y, k) \equiv \left.\dfrac{d^k p(x)}{dx^k}\right|_{x=y} = C^k\, q_0$
```

Table 1: Coefficients for the evaluation of the k-th derivative of a finite series of Jacobi, Gegenbauer, Hermite and generalized Laguerre polynomials.

$\mathcal{P}$	Jacobi	Gegenbauer	Hermite		Laguerre
	$P_n^{(\alpha,\beta)}$	C_n^{λ}	H_n	He_n	$L_n^{(\alpha)}$
D	$[-1, 1]$	$[-1, 1]$	$\mathbb{R}$	$\mathbb{R}$	$(0, +\infty)$
C^k	$\dfrac{1}{2^k}$	$2^k\,(\lambda)_k$	2^k	1	$(-1)^k$
$A_j^{c,k}$	$(j+k+\alpha+\beta+1)_k$	1	$(j-k+1)_k$	$(j-k+1)_k$	1
$A_j^{1,k}(y)$	$\dfrac{2j+1+2k+\alpha+\beta}{(2j+2)(j+1+\alpha+\beta+2k)} \times\left\{(2j+2+2k+\alpha+\beta)\,y + \dfrac{(\alpha+k)^2-(\beta+k)^2}{2j+2k+\alpha+\beta}\right\}$	$\dfrac{2(j+k+\lambda)}{j+1}\,y$	$2y$	y	$\dfrac{-y}{j+1} + \dfrac{2j+\alpha+k+1}{j+1}$
$A_j^{2,k}$	$-\dfrac{(j+1+\alpha+k)(j+1+\beta+k)}{(j+2)(j+2+2k+\alpha+\beta)} \times\dfrac{(2j+4+2k+\alpha+\beta)}{(2j+2+2k+\alpha+\beta)}$	$-\dfrac{(j+2\lambda+2k)}{j+2}$	$-2(j+1)$	$-(j+1)$	$-\dfrac{j+1+\alpha+k}{j+2}$

The BCS-algorithm provides a useful algorithm for the evaluation of orthogonal polynomial series and their derivatives, but it is important to remark that the numerical evaluation of derivatives is a badly conditioned problem. Therefore, the appearance of rounding errors may deteriorate the evaluation of high order derivatives. That is, although the BCS-algorithm is, theoretically, valid for any derivative order, in some practical applications this may not be the case (in any case, useful up to the standard order of derivatives, for instance for collocation methods). A quite useful theoretical tool is to equip the BCS-algorithm with error bounds computed at the same time as the algorithm itself which provides *a posteriori* error bounds of the algorithm. This is reached by means of the so-called *running-error analysis* [31]. A *running-error bound* of the BCS-algorithm, obtained adapting the results of [24, 25], for the evaluation of the k-th derivative of the polynomial series $p(x) = \sum_{j=0}^{n} c_j\, p_j(x)$, is given by

$$\left|\widehat{p}^{(k)}(x) - p^{(k)}(x)\right| \le u \cdot C^k (x_0 - 2\left|\widehat{q}_0\right|) + O(u^2), \tag{3}$$

where x_0 is derived from

$$x_{n-k+1} = y_{n-k+2} = y_{n-k+1} = 0, \tag{4}$$

$$\left.\begin{aligned} z_j &= x_{j+1}\left|\widehat{A}_j^{1,k}(x)\right| + y_{j+2}\left|\widehat{A}_j^{2,k}\right| + (2+n_c)\left|\widehat{A}_j^{c,k} c_{j+k}\right|, \\ x_j &= z_j + (3+n_1)\left|\widehat{q}_j\right|, \\ y_j &= z_j + (2+n_2)\left|\widehat{q}_j\right|, \end{aligned}\right\} \quad \text{for} \quad j = n-k, \ldots, 0, \tag{5}$$

Table 2: Coefficients for the evaluation of the k-th derivative of a finite series of Chebyshev polynomials of the first and second kind and Legendre polynomials (denoting $\delta_{j,0}$ the Kronecker delta).

$\mathcal{P}$	Chebyshev (1st) T_n			Chebyshev (2nd) U_n			Legendre P_n	
D	$[-1, 1]$			$[-1, 1]$			$[-1, 1]$	
	$k = 0$	$k = 1$	k	$k = 0$	$k = 1$	k	$k = 0$	k
C^k	1	1	$2^{k-1}(k-1)!$	1	2	$2^k k!$	1	$(2k-1)!!$
$A_j^{c,k}$	1	$(j+1)$	$(j+k)$	1	1	1	1	1
$A_j^{1,k}(y)$	$(2-\delta_{j,0})\,y$	$2y$	$\dfrac{2(j+k)}{j+1}\,y$	$2y$	$\dfrac{2j+4}{j+1}\,y$	$\dfrac{2(j+k+1)}{j+1}\,y$	$\dfrac{2j+1}{j+1}\,y$	$\dfrac{2j+2k+1}{j+1}\,y$
$A_j^{2,k}$	$-\left(1-\dfrac{\delta_{j,0}}{2}\right)$	-1	$-\dfrac{j+2k}{j+2}$	-1	$-\dfrac{j+4}{j+2}$	$-\dfrac{j+2k+2}{j+2}$	$-\dfrac{j+1}{j+2}$	$-\dfrac{(j+1+2k)}{j+2}$

being $\widehat{p}^{(k)}(x)$ and $\widehat{q}_i$ the computed values of the BCS-algorithm (Algorithm 1), and under the assumption that the evaluation of the coefficients satisfies $\widehat{A_j^{c,k}} \approx A_j^{c,k}\{1 + (2 + n_c)u\}$, $\widehat{A_j^{1,k}}(x) \approx A_j^{1,k}(x)\{1 + (3 + n_1)u\}$ and $\widehat{A_j^{2,k}} \approx A_j^{2,k}\{1 + (2 + n_2)u\}$, where u denotes the round-off unit of the computer and the terms n_c, n_1 and n_2 are given by

	Jacobi		Gegenbauer	Chebyshev (1st)		Chebyshev (2nd)		Legendre	Hermite	Laguerre
	$(k = 0)$	$(k \geq 1)$		$(k = 0, 1)$	$(k \geq 2)$	$(k = 0)$	$(k \geq 1)$			
n_c	0	$k + 1$	0	0	0	0	0	0	0	0
n_1	6	7	3	0	2	0	2	2	0	3
n_2	5	5	2	0	1	0	1	1	0	2

The interesting fact of having a running-error bound is that it provides us with information about the quality of the numerical evaluation of the series and of the derivatives, something that can be crucial in some practical applications. Note that this kind of theoretical bounds is very useful in detecting ill-conditioned problems in cases of low degree polynomials, $n \lesssim O(10^2)$ and when the absolute values of the family of polynomials are $\ll 1/\varepsilon$ (in this case the bounds are usually quite sharp), but for high degree polynomials the bounds are too pessimistic (as any theoretical bound) due to the use of recurrences with absolute values that give rise to a constant increment in size of the bound.

3. Accurate algorithms: Error-Free Transformation techniques

As commented above, the running-error bound (3) provides information about the accuracy of the evaluation process, and so, an interesting question is what to do when the accuracy is low.

In order to help the stability study of polynomial evaluation, in [32] we introduced a general condition number for any polynomial basis that satisfies a homogeneous linear recurrence (like power, Bernstein and any orthogonal polynomial basis). Now we can state a theoretical result about the numerical stability of the BCS-algorithm developed in previous section. Using (Theorem 3.5, [32]) we obtain that if $p(x) = \sum_{j=0}^{n} c_j p_j(x)$, expressed in the orthogonal polynomial basis $\Phi = \{p_j(x)\}$ satisfying (1), and $\widehat{p}^{(k)}(x)$ is the value of the k-th derivative computed by the BCS-algorithm, then, up to first order in u,

$$|\widehat{p}^{(k)}(x) - p^{(k)}(x)| \leq 6(n+1) \cdot u \cdot \sum_{j=0}^{n} |c_j| (p_j^{(k)})^{\sharp}(x) \equiv 6(n+1) \cdot u \cdot S_{\Phi}(p^{(k)}(x)) \equiv O(u) \cdot S_{\Phi}(p^{(k)}(x)), \tag{6}$$

where $\{(p_j^{(k)})^\sharp(x)\}$ is the basis of the absolute polynomials associated with the derivative basis $\{p_j^{(k)}(x)\}$ defined by

$$(p_0^{(k)})^\sharp(x) = 1, \quad (p_1^{(k)})^\sharp(x) = \left|A_1^{1,k}(x)\right|, \quad (p_j^{(k)})^\sharp(x) = \left|A_j^{1,k}(x)\,(p_{j+1}^{(k)})^\sharp(x)\right| + \left|A_j^{2,k}\,(p_{j+2}^{(k)})^\sharp(x)\right|, \quad j \geq 2.$$

We remark that the term $S_\Phi(p^{(k)}(x)) := \sum_{j=0}^{n} |c_j| (p_j^{(k)})^\sharp(x)$, is the generalized condition number for polynomial evaluation introduced in [32].

The theoretical error bound (6) states that the evaluation algorithms are stable but note that the condition number can grows significatively for some families of orthogonal polynomials deteriorating the evaluation process. For instance, it is well known that for long series (n large) and large derivative order k the accuracy may be quite poor [25]. Therefore, several techniques for reducing the rounding errors are recommended, specially for the evaluation of long series or high derivatives, as commented. The recent development of some families of more stable algorithms, which are called *compensated algorithms* [26], is based on the paper [33] about *Error-Free Transformations* (EFT). For a pair of floating-point numbers $a, b \in \mathbb{F}$, when no underflow occurs, there exists a floating-point number y satisfying $a \circ b = x + y$, where $x = \mathrm{fl}(a \circ b)$ and $\circ \in \{+, -, \times\}$. Then the transformation $(a, b) \longrightarrow (x, y)$ is regarded as an EFT. The error-free transformation algorithms of the sum and product of two floating-point numbers used later in this paper are the `TwoSum` algorithm by Knuth [34] and the `TwoProd` algorithm by Dekker [35], respectively. The `TwoSum` and `TwoProd` algorithms (see [33]) satisfy the properties

$$\begin{aligned} &[x, y] = \texttt{TwoSum}(a, b), \quad x = \mathrm{fl}(a + b), \quad x + y = a + b, \quad |y| \leq u|x|, \quad |y| \leq u|a + b|, \\ &[x, y] = \texttt{TwoProd}(a, b), \quad x = \mathrm{fl}(a \times b), \quad x + y = a \times b, \quad |y| \leq u|x|, \quad |y| \leq u|a \times b|, \end{aligned}$$

where $\mathrm{fl}(a + b)$ and $\mathrm{fl}(a \times b)$ stand for the computed result of the corresponding operations.

The use of the EFT techniques allows to relegate the influence of the condition number up to second order in u, and thus now the forward error bound is [32, 36]

$$|\widehat{p}^{(k)}(x) - p^{(k)}(x)| \leq u \cdot |p^{(k)}(x)| + O(u^2) \cdot S_\Phi(p^{(k)}(x)). \tag{7}$$

The great advantage of bound (7) with respect to (6) is that now the condition number is multiplied by u^2, and therefore, in most of the situations we may expect a highly accurate evaluation, and only when the condition number is very large we may have inaccurate results.

As an example of two of the algorithms included in the library ORTHOPOLY, we show the standard Clenshaw algorithm for evaluating Chebyshev ($T_i(x)$) series and the corresponding compensated Clenshaw algorithm [36].

Algorithm 2. *Clenshaw algorithm to evaluate finite Chebyshev series*

```
function Clenshaw($p, x$)
    $b_{n+2} = b_{n+1} = 0$
    for $j = n : -1 : 1$
        $b_j = 2xb_{j+1} - b_{j+2} + a_j$
    end
    Clenshaw($p, x$) $\equiv p(x) \equiv b_0 = xb_1 - b_2 + a_0$
```

Algorithm 3. *Compensated Clenshaw algorithm to evaluate accurately finite Chebyshev series*

```
function CompClenshaw($p, x$)
    $\hat{b}_{n+2} = \hat{b}_{n+1} = 0$
    $\epsilon b_{n+2} = \epsilon b_{n+1} = 0$
    for $j = n : -1 : 1$
        $[s, \pi_j]$=TwoProd($\hat{b}_{j+1}, 2x$)
        $[v, \sigma_j]$=TwoSum($s, -\hat{b}_{j+2}$)
        $[\hat{b}_j, \beta_j]$=TwoSum($v, a_j$)
        $\hat{w}_j = \pi_j + \sigma_j + \beta_j$
        $\epsilon b_j = 2x \times \epsilon b_{j+1} - \epsilon b_{j+2} + \hat{w}_j$
    end
    $[s, \pi_0]$=TwoProd($\hat{b}_1, x$)
```

```
$[v, \sigma_0]$=TwoSum$(s, -\hat{b}_2)$
$[\hat{b}_0, \beta_0]$=TwoSum$(v, a_0)$
$\hat{w}_0 = \pi_0 + \sigma_0 + \beta_0$
$\epsilon b_0 = x \times \epsilon b_1 - \epsilon b_2 + \hat{w}_0$
CompClenshaw$(p, x) \equiv \hat{b}_0 + \epsilon b_0$
```

4. Overview of the software structure

The global structure of the ORTHOPOLY library in the C version consists on 3 folders with the `.c` programs (`src` folder), the header `.h` files (`include` folder) and a basic example (`example` folder), plus a `readme` file with an example of the compilation of a subroutine. For each family of orthogonal polynomials there are 18 subroutines, three versions (standard, compensated and “accurate”) of each option, for the different evaluation algorithms:

```
  -.1 #####Val:           evaluates a series of ##### polynomials
  -.2 Comp#####Val:       evaluates a series of ##### polynomials
                            with compensated method (double floating point output)
  -.3 Acc#####Val:        evaluates a series of ##### polynomials
                            with compensated method (double-double output)
  -.4 #####ValwErr:       evaluates a series of ##### polynomials
                            with running-error bound
  -.5 Comp#####ValwErr:   evaluates a series of ##### polynomials
                            with compensated method (double floating point output)
                            with running-error bound
  -.6 Acc#####ValwErr:    evaluates a series of ##### polynomials
                            with compensated method (double-double output)
                            with running-error bound
--------------------------------------------------------------------------------------
  -.7  #####Der:          evaluates the first derivative of a series of ##### polynomials
  -.8  Comp#####Der:      evaluates the first derivative of a series of ##### polynomials
                            with compensated method (double floating point output)
  -.9  Acc#####Der:       evaluates the first derivative of a series of ##### polynomials
                            with compensated method (double-double output)
  -.10 #####DerwErr:      evaluates the first derivative of a series of ##### polynomials
                            with running-error bound
  -.11 Comp#####DerwErr:  evaluates the first derivative of a series of ##### polynomials
                            with compensated method (double floating point output)
                            with running-error bound
  -.12 Acc#####DerwErr:   evaluates the first derivative of a series of ##### polynomials
                            with compensated method (double-double output)
                            with running-error bound
--------------------------------------------------------------------------------------
  -.13 #####DerK:         evaluates the k-th derivative of a series of ##### polynomials
  -.14 Comp#####DerK:     evaluates the k-th derivative of a series of ##### polynomials
                            with compensated method (double floating point output)
  -.15 Acc#####DerK:      evaluates the k-th derivative of a series of ##### polynomials
                            with compensated method (double-double output)
  -.16 #####DerKwErr:     evaluates the k-th derivative of a series of ##### polynomials
                            with running-error bound
  -.17 Comp#####DerKwErr: evaluates the k-th derivative of a series of ##### polynomials
                            with compensated method (double floating point output)
                            with running-error bound
  -.18 Acc#####DerKwErr:  evaluates the k-th derivative of a series of ##### polynomials
```

```
                                  with compensated method (double-double output)
                                  with running-error bound
```

where ##### stands for the particular family of orthogonal polynomials (T_n Cheb1, U_n Cheb2, P_n Legen, C_n^λ Gegen, $P_n^{(\alpha,\beta)}$ Jacob, $L_n^{(\alpha)}$ Lague, He_n Herm1, H_n Herm2).

The generic description of the inputs of the subroutines in C is given by

```
subroutine_name(double *P,      % P list of coefficients of the series
          unsigned int n,       % n order of the series
          double x,             % x point of evaluation
          double PARAMETER,     % PARAMETER of the family of orthogonal polynomials
          unsigned int k,       % k order of derivation (when k>1)
          double * runerrbound)% pointer to the running-error bound (when used)
```

Note that the PARAMETER has to be given just for the Gegenbauer, Jacobi and Laguerre families. And the last three inputs appear just in some subroutines, whereas the first three ones are common in all the subroutines.

In the Matlab version there are just 12 functions for each family (in a particular folder for each one) because now there is only one compensated version of the algorithm (that is, there is not a Comp##### and a Acc##### version, just one, that corresponds to the Comp##### case). The description of the inputs and outputs is detailed on each function m-file, and the file startOrthoPoly.m initializes the paths to be able to use the functions.

At this point it is important to remark that the compensated algorithms reach their best accurate performance in C with the Acc##### version as the use of the double-double output permits to correct some extra digits when all the digits in double precision are correct but the last one (in the case of the Comp##### version this is not possible). Generically, the use of the running-error bound is recommended mainly for low degree polynomials as in that case it detects ill-conditioned situations and provides sharp bounds in some cases. In the case of high-degree polynomials (or medium degree for families of polynomials with high absolute values at some points) the error bounds provide highly overestimated bounds (as in any error bound in literature).

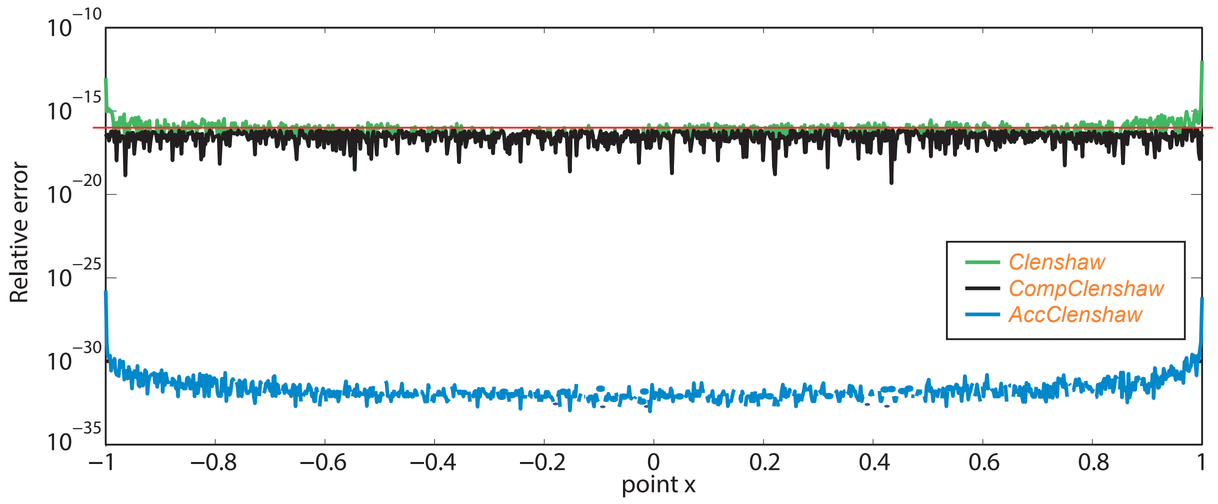

Figure 1: Relative errors in the evaluation of a Gegenbauer ($\lambda = 1/10$) series of degree $n = 1000$ with random coefficients obtained with the three different subroutines in C for the evaluation algorithms (GegenVal, CompGegenVal and AccGegenVal).

In order to show the use of the three versions of the programs in C (in Matlab there are just two, as explained before), we present in Figure 1 the relative errors in the evaluation of a Gegenbauer ($\lambda = 1/10$) series of degree $n = 1000$ with random coefficients obtained with the three different subroutines in C for the evaluation algorithms (GegenVal, CompGegenVal and AccGegenVal). We observe that in all cases the algorithm gives a quite accurate result but the standard one (GegenVal) near the ends of the interval. What is important to remark is that the compensated version

(`CompGegenVal`) provides a relative error that is always on the level of the round-off unit of the computer (the standard one has errors 1000 times bigger at some points), and the accurate version (`AccGegenVal`) uses the compensated algorithm but it provides a double-double output that permits to obtain extra precision digits (but in this case the use is more complicated as we have to link both outputs, a standard user will use the `Comp#####` version).

5. Numerical tests

In this section we intend to show some special features of the ORTHOPOLY library. To that goal we present some numerical tests using the standard version of the algorithms and the compensated ones, remarking that the compensated versions permit to obtain highly accurate results.

All our experiments have been performed using IEEE-754 double precision with round-off unit $u \approx 1.16 \times 10^{-16}$. The "more precise" values in the comparisons are taken by using quadruple precision by means of the double-double arithmetic [37, 38] and using the coefficients in double precision (extra digits are 0s).

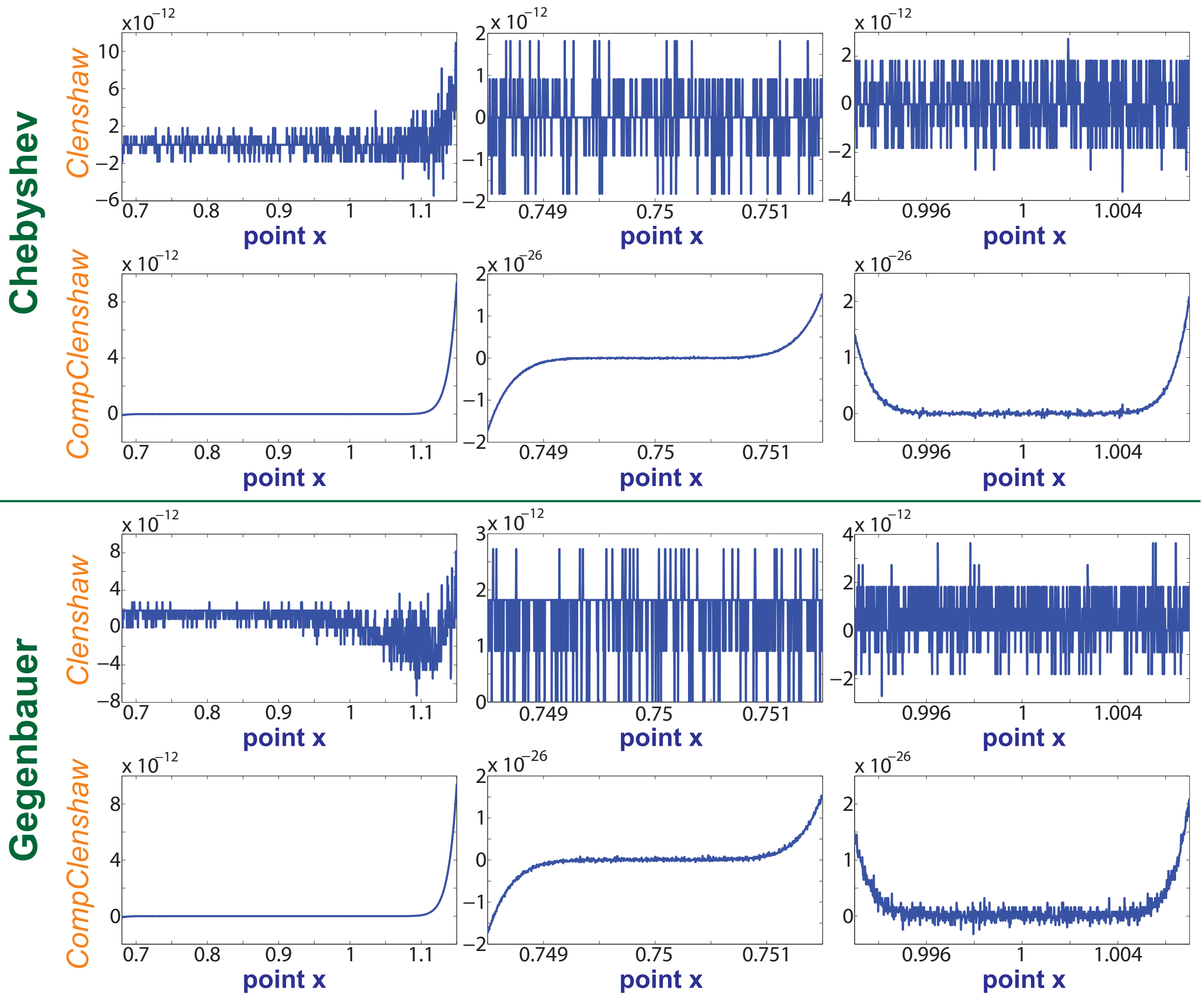

Figure 2: Evaluation of the polynomial $p(x) = (x - 0.75)^7(x - 1)^{10}$ in Chebyshev (of the first kind) and Gegenbauer ($\lambda = 1/10$) series form in the neighborhood of its multiple roots, using Clenshaw (up) and compensated Clenshaw (down) algorithms.

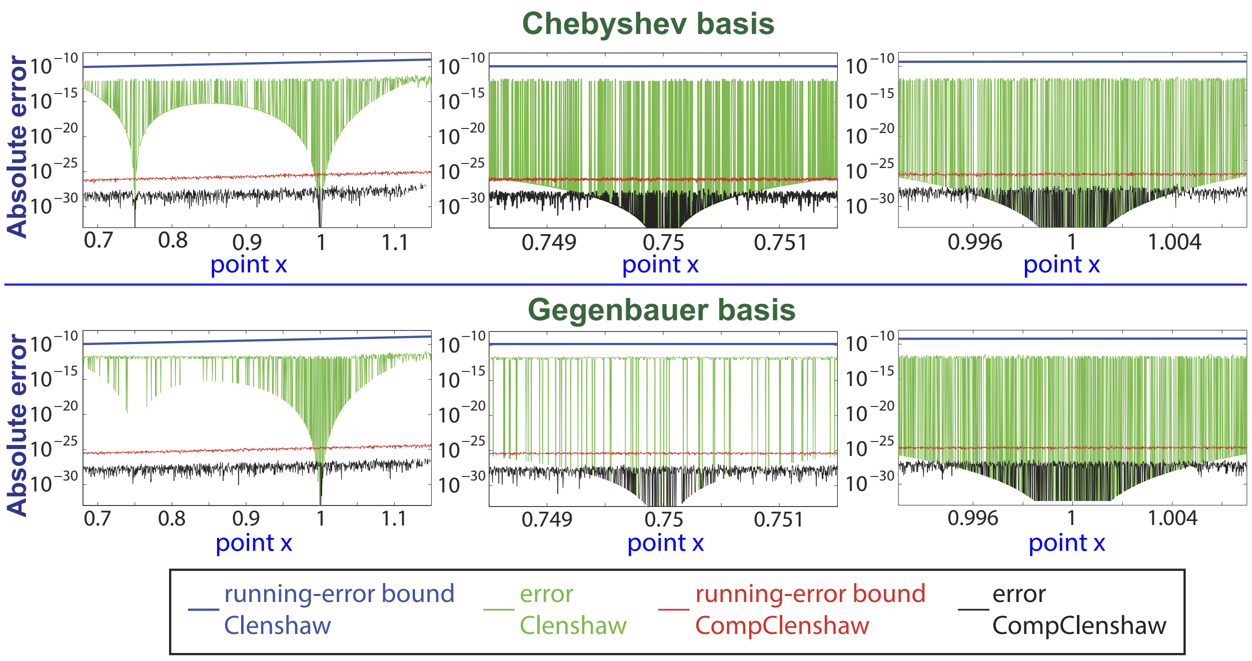

Figure 3: Comparison among the running-error bounds and the real errors for the Clenshaw and compensated Clenshaw algorithms in the evaluation of the polynomial $p(x) = (x - 0.75)^7(x - 1)^{10}$ in Chebyshev (of the first kind) and Gegenbauer ($\lambda = 1/10$) series form in the neighborhood of its multiple roots.

As first problem we consider the evaluation of an ill-conditioned polynomial, in our case the evaluation in the neighborhood of its multiple roots 0.75 and 1, of the polynomial $p(x) = (x-0.75)^7(x-1)^{10}$ written in Chebyshev form $p(x) = \sum_{i=0}^{17} a_i T_i(x)$ and in Gegenbauer form $p(x) = \sum_{i=0}^{17} a_i C_i^{1/10}(x)$ ($\lambda = 1/10$). In order to compute the relative error, we use the MATLAB Symbolic Toolbox to accurately evaluate the polynomial. The conversion algorithms from power series representation into Chebyshev and Gegenbauer representation may be obtained from [39]. Figure 2 presents the evaluation of both representations of the polynomial for 1000 equally spaced points in the intervals $[0.68, 1.15]$, $[0.7485, 0.7515]$ and $[0.993, 1.007]$. It is clear that our compensated Clenshaw algorithm [36] gives a much more smooth drawing than the original Clenshaw algorithm in both orthogonal polynomial basis. Moreover, the relative error is always (except for x too close to $p(x) = 0$, that is, $x \simeq 0.75$ and 1) of the order of the round-off unit u. Thus, the effective behavior of the accurate polynomial evaluation is obvious. Note that this can be crucial in algorithms for locating zeros of polynomials in floating-point arithmetic, like the Newton's method, because the oscillations, like the ones presented on the top figures, can make impossible to obtain an accurate result for ill conditioned polynomials. The evaluations have been done using the subroutines `Cheb1ValwErr`, `CompCheb1ValwErr`, `GegenValwErr` and `CompGegenValwErr` of the ORTHOPOLY library.

We remark that an important point of the ORTHOPOLY library is the option to have, at the same time as the evaluation of the polynomial series and their derivatives, a running-error bound that provides us with some information about the accuracy of the numerical result. To test that goal we present in Figure 3 a comparison of the running-error bounds (3) and the real errors in the evaluation done in the tests of Figure 2. In Figure 3 we observe that the bounds provided by (3), and implemented in the ORTHOPOLY library, give us a quite useful information about the real errors for both, the standard and compensated algorithms. The evaluations and the running-error bounds have been done using the same subroutines as in Figure 2. As above commented, the running-error bounds are quite useful in detecting inaccurate evaluations in case of ill-conditioned problems, as it is the case of this numerical test. We remark that when the polynomial families have large absolute values at some points (as occurs, for instance, when using Jacobi or Gegenbauer polynomials with parameters higher than 1) or high-degree polynomials, the error bounds provide highly overestimated bounds.

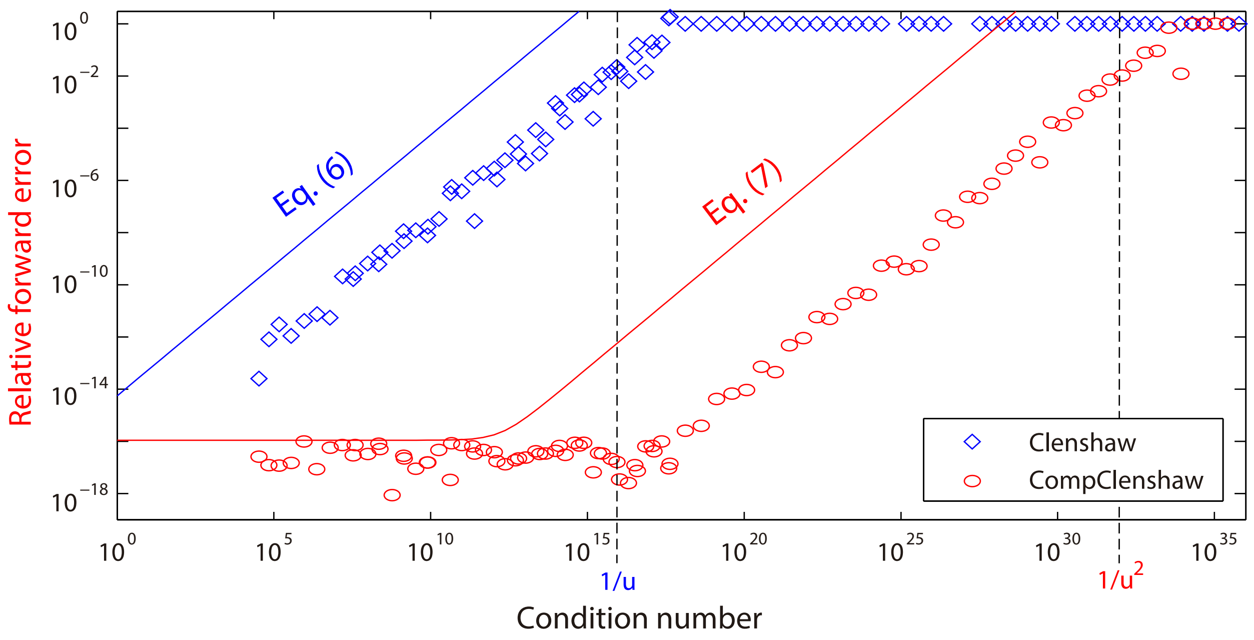

Figure 4: Relative accuracy of the evaluation of $p(x) = (x - 0.75)^7(x - 1)^{10}$ represented in Chebyshev form with respect to the relative condition number $|S_T(p(x))/p(x)|$, and the theoretical bounds given by Equations 6 and 7.

In the next test, we focus on the forward error bound of our compensated Clenshaw algorithm, and on the usefulness of that algorithm (that is, if it always provides highly accurate results or not). In the case of the evaluation of a polynomial series, it is known that the closer the argument is to the root, the more ill-conditioned the evaluation is. Therefore, we evaluate the Chebyshev form of the polynomial $p(x) = (x-0.75)^7(x-1)^{10}$ in terms of Chebyshev polynomials of the first kind at the floating-point entries whose corresponding relative condition number $|S_T(p(x))/p(x)|$ vary from 10^2 to 10^{35}. These floating-point entries are generated by means of formulas $x = 0.75 - (1.03)^{(2i-85)}$, for $i = 40 : -1 : 1$ and $x = 0.75 - (1.13)^{(i-85)}$, for $i = 80 : -1 : 1$. The results are reported on Figure 4. As we can see, the compensated Clenshaw algorithm exhibits the expected behavior, that is, when the relative condition number is smaller than $1/u$, the relative error is equal to or smaller than u. This relative error increases linearly for the relative condition number between $1/u$ and $1/u^2$. Note that the theoretical bound (7) is very sharp when $|S_T(p(x))/p(x)| < 1/u$ for the compensated Clenshaw algorithm because in this case the relative error bound is essentially the round-off unit. When the relative condition number grows, the bound is not so sharp although it gives precise qualitative information, as the bound (6) for the Clenshaw algorithm. Also we note, as expected, that when the condition number is very high ($> 1/u^2$), then we cannot expect any precision at all (note that the standard Clenshaw algorithm is not useful much before ($> 1/u$)). We remark that this situation is given just for very large series and taking large values of the parameters in some families of orthogonal polynomials (because in that case the value of the polynomials of the family can be very large) or very close to zeros of the finite polynomial series (as in this case the global value is too small, as in the case of Figure 4).

Another interesting numerical test is related to maintain the quality of the evaluation of the Chebyshev series along all the interval of definition D when the degree of the polynomial is high, that is, long polynomial series. Note that now usually we do not face to ill-conditioned evaluations (unless, of course, we intend also to evaluate near a multiple root as before) but we have condition problems due to the high degree of the polynomials. These kind of series are quite used in collocation or pseudospectral methods for solving numerically PDEs. It is well known that in this situation the evaluation algorithms may have problems near the ends of the interval [40]. Therefore, we have also considered the cases of degree $n = 10^2$, 10^3, 10^4 and 10^5 for the evaluation of the Chebyshev, Gegenbauer ($\lambda = 1/10$) and Laguerre ($\alpha = 2$) series with uniformly random coefficients in $(0, 1)$. Figure 5 illustrates that the larger the degree n, the lower accuracy the Clenshaw-Smith algorithm has, which is consistent with the theoretical error bound (6). But

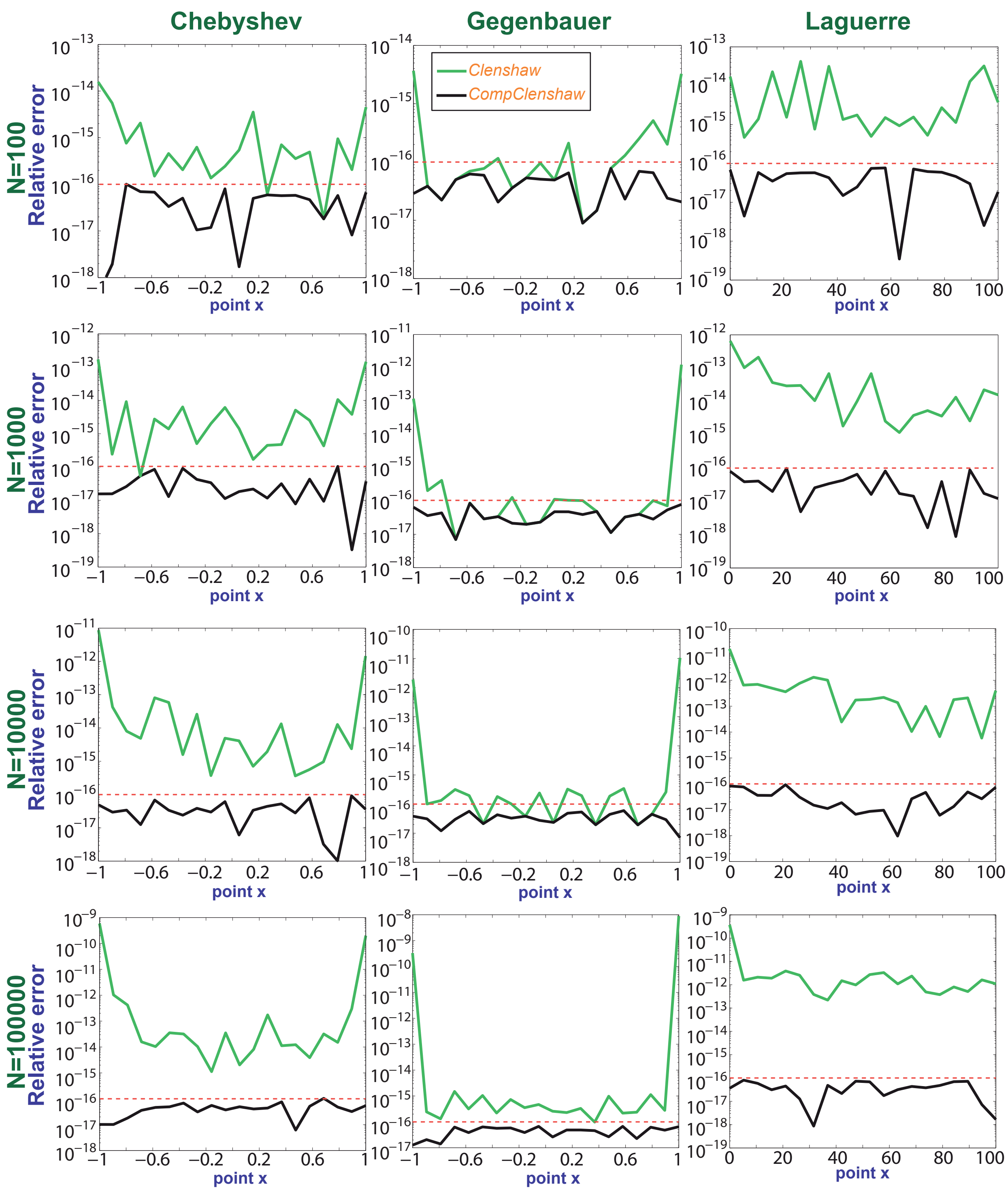

Figure 5: Relative errors in the evaluation of Chebyshev, Gegenbauer ($\lambda = 1/10$) and Laguerre ($\alpha = 2$) series of degree $n = 10^2, 10^3, 10^4$ and 10^5 with random coefficients.

the compensated Clenshaw-Smith algorithm has always high accuracy in any case, which is theoretically illustrated with the theoretical error bound (7) as the condition number is now relegated to second order in the round-off unit. The evaluations have been done using the subroutines `Cheb1Val`, `CompCheb1Val`, `GegenVal`, `CompGegenVal`, `LagueVal` and `CompLagueVal` of the ORTHOPOLY library.

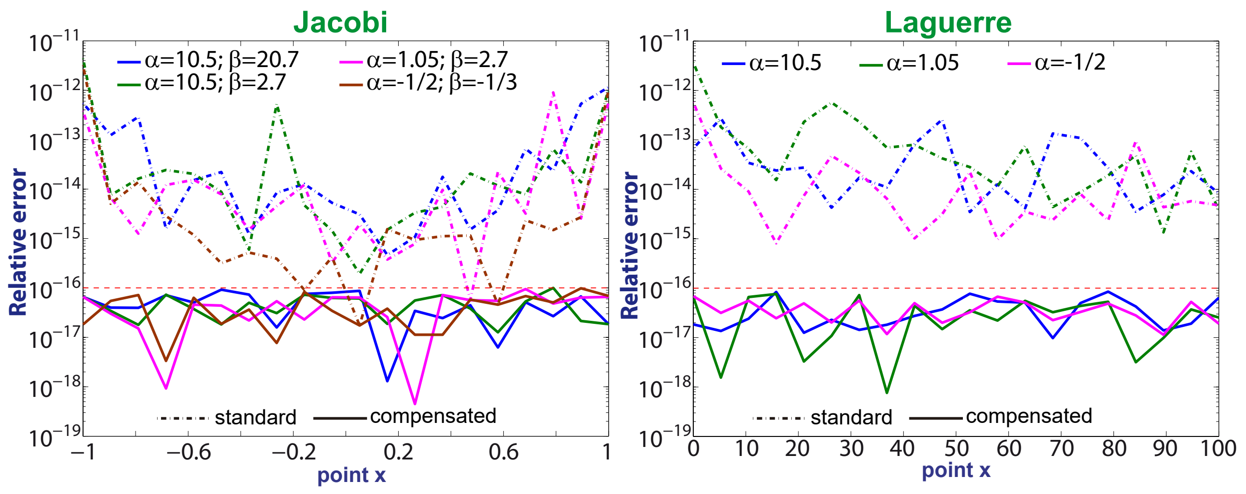

Figure 6: Relative errors in the evaluation of Jacobi and Laguerre series of degree $n = 1000$ with random coefficients for different values of the parameters of the family of polynomials depending on the point of evaluation x.

As we are working with families of polynomials with parameters in their definition, it is interesting to see the robustness of the algorithms with respect to them. In Figure 6 we show the relative errors in the evaluation of Jacobi and Laguerre series of degree $n = 1000$ with random coefficients for different values of the parameters of the family of polynomials: Jacobi $\alpha = -1/2, \beta = -1/3$; $\alpha = 1.05, \beta = 2.7$; $\alpha = 10.5, \beta = 2.7$; $\alpha = 10.5, \beta = 20.7$, and Laguerre $\alpha = -1/2$; $\alpha = 1.05$; $\alpha = 10.5$. Similarly, in Figure 7 we show the relative errors of the algorithms but now as a function of the value of the parameter α of the family of Jacobi and Laguerre polynomials. In this figure we have considered 20 values of variable x uniformly distributed in the interval considered for each family ($[-1, 1]$ for Jacobi and $[0, 100]$ for Laguerre), and in the graph the maximum relative error for each value of α is represented. From the tests we observe, as in previous tests, a good global behavior of the standard Clenshaw-Smith algorithm. Moreover, the compensated Clenshaw-Smith algorithm has always very high relative accuracy (a relative error of the order of the round-off unit), also in cases where the precision of the standard method is around 10^{-10}, although the absolute error can be high because these polynomials may have a very large value (for instance, the Jacobi polynomial of degree 1000 with $\alpha = 10.5, \beta = 20.7$ reaches a maximum absolute value of the order 10^{42}).

Thus, the compensated Clenshaw-Smith algorithm (and therefore, the compensated subroutines of the ORTHOPOLY library) can be used to maintain a global high precision along all the interval of evaluation in case of high degree polynomials, taking into account that in several real situations we need more precision precisely at the ends of the interval as these points are the connecting points among different intervals.

A remarkable application of the ORTHOPOLY library is in the evaluation of derivatives of polynomial series. In this case the algorithm used is the BCS-algorithm and the behavior of the standard and compensated versions is presented in Figure 8, where we show the relative errors in the evaluation of the derivatives of Chebyshev, Gegenbauer and Laguerre series of degree $n = 100$ with coefficients $c_i = r_i/i^4$, being r_i a uniformly random number in $(-2, 2)$. From the figures it is clear that the behavior of the algorithms is quite good, and specially remarkable is the behaviour of the compensated version that allows to maintain an error of the order of the round-off unit of the computer (the red horizontal line in the plots) also for the 4th derivative. We check the error with respect to a quadruple precision evaluation of the polynomial series. The evaluations have been done using the subroutines `Cheb1Der`, `Cheb1DerK`, `CompCheb1Der`, `CompCheb1DerK`, `GegenDer`, `GegenDerK`, `CompGegenDer`, `CompGegenDerK`, `LagueDer`, `LagueDerK`, `CompLagueDer` and `CompLagueDerK` of the ORTHOPOLY library. We

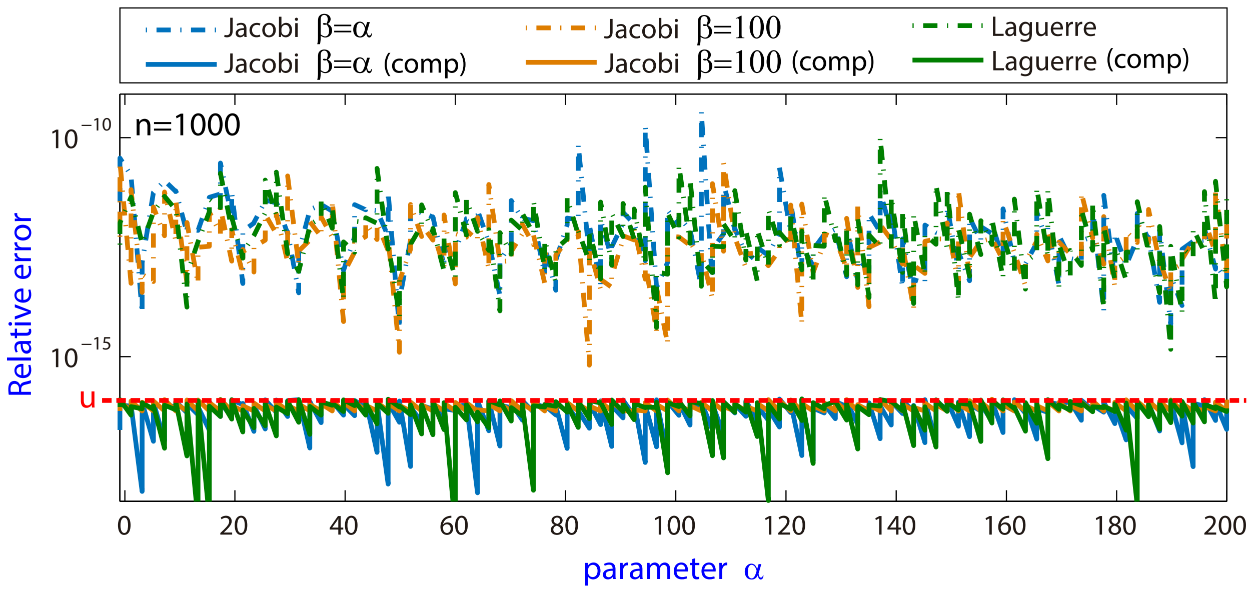

Figure 7: Relative errors in the evaluation of several Jacobi and Laguerre series of degree $n = 1000$ with random coefficients depending on the parameter α of the family of polynomials.

point out the different interval of evaluation of the different polynomial series as the Laguerre orthogonal polynomials are defined in the interval $(0, +\infty)$.

Finally, we show in Figure 9 some CPU time measurements of some of the algorithms performed with the C version of the ORTHOPOLY library evaluating polynomials series with random coefficients of several degrees n and their first derivative. All the tests have been done on a workstation Xeon Haswell E5-2698 V3 (2.3 GHz), under Linux using standard gcc compiler. The special version of the algorithms for just the evaluation of the Chebyshev series is the fastest one because in this case the recurrence is specially simple (this is quite important in practical applications in spectral and collocations methods). From the pictures it is also important to remark the small increment in CPU time of using the versions of the algorithms that, at the same time of the evaluation itself, give the running-error bound. Therefore, to have some information of the accuracy of the results is not “expensive”, and so it can be advisable. The highly accurate compensated algorithms are more expensive, of course, but in any case the computer time is not really high, taking into account that the compensated algorithms may provide results (in most of the situations) with all the significative numbers exact. These numerical tests show that the algorithms used in the ORTHOPOLY library permit to evaluate orthogonal polynomial series, and their derivatives, accurately and in a reliable CPU time.

6. Application example: Hydrogen atom

In this section we show a simple direct application of the ORTHOPOLY library in a physical problem. We consider the radial part of the Schrödinger equation in M dimensions [5, 41, 42]

$$\left[-\frac{\mathrm{d}^2}{\mathrm{d}r^2} - \frac{M-1}{r}\frac{\mathrm{d}}{\mathrm{d}r} + \frac{l(l+M-2)}{r^2} + V(r)\right]\mathcal{R}^{(M)}_{n,l}(r) = E^{(M)}_{n,l}\mathcal{R}^{(M)}_{n,l}(r), \tag{8}$$

where r denotes the radial distance, n and l are the radial and angular quantum numbers of the energy eigenvalues $E^{(M)}_{n,l}$ and the corresponding radial wave functions $\mathcal{R}^{(M)}_{n,l}(r)$. The most common situation is the three-dimensional case $M = 3$ and making use of the transformation $R_{n,l}(r) = r\mathcal{R}^{(3)}_{n,l}(r)$, we obtain [42]

$$\left[-\frac{\mathrm{d}^2}{\mathrm{d}r^2} + \frac{l(l+1)}{r^2} + V(r)\right]R_{n,l}(r) = E_{n,l}R_{n,l}(r). \tag{9}$$

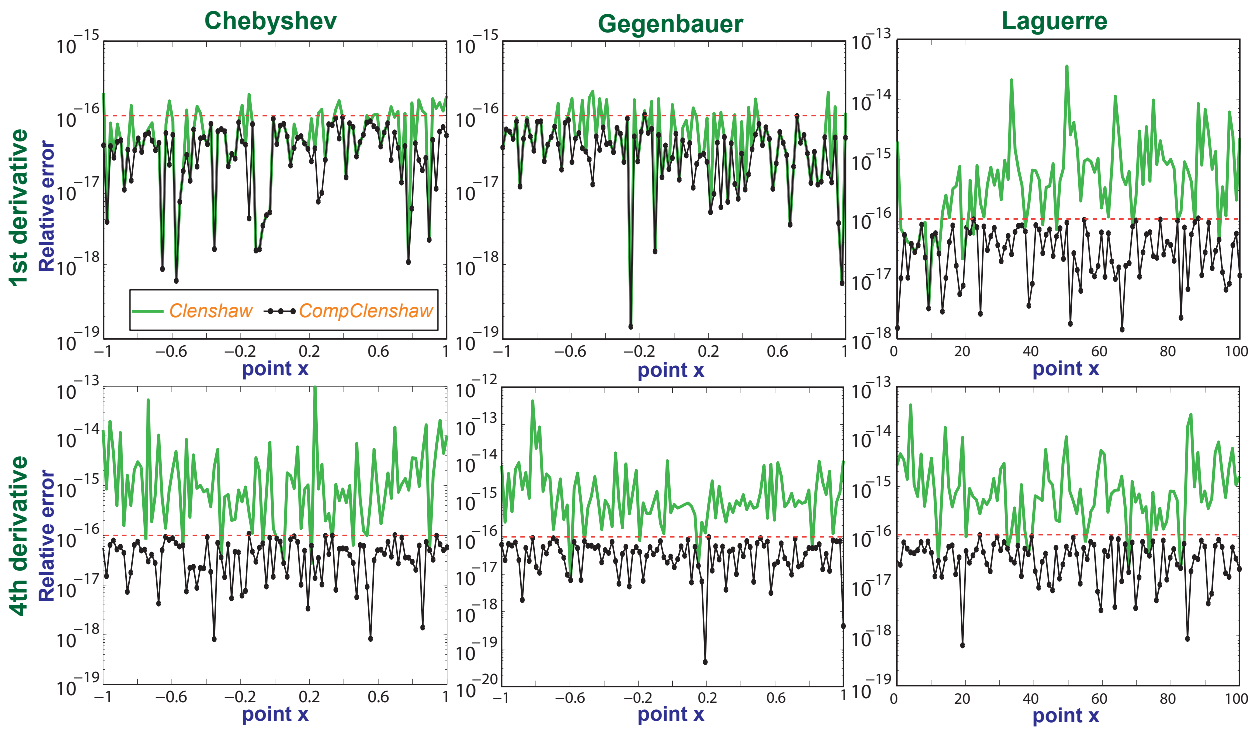

Figure 8: Relative errors in the evaluation of the derivatives of Chebyshev, Gegenbauer ($\lambda = 1/10$) and Laguerre ($\alpha = 2$) series of degree $n = 100$.

Considering now the quantum Coulomb mechanical potential $V(r) = -1/r$ we have the radial equation of the non-relativistic Schrödinger equation for an hydrogenic ion

$$\left[-\frac{\mathrm{d}^2}{\mathrm{d}r^2} + \frac{l(l+1)}{r^2} - \frac{1}{r}\right] R_{n,l}(r) = E_{n,l} R_{n,l}(r), \qquad r \in [0, +\infty). \tag{10}$$

Therefore, the problem is to solve the equation (10) to obtain the radial wave functions $R_{n,l}(r)$ and the allowed energies $E_{n,l}$. This problem admits a solution [41] in terms of the generalized Laguerre polynomials $L_n^{(\alpha)}$, that in the particular case of $l = 0$ is given by

$$E_{n,0} = -\frac{1}{2n^2}, \qquad R_{n,0}(r) = r \exp(-r/n) L_{n-1}^{(1)}(2r/n), \qquad n \geq 1. \tag{11}$$

So, we will use this particular case as example as we will numerically solve equation (10) using the Laguerre pseudospectral method [4, 5, 10] (due to the radial interval $[0, +\infty)$) and we will compare with the exact solution (11). The Laguerre pseudospectral method [10] solves the equation (10) by approximating the solution by a finite series of Laguerre polynomials L_j

$$R_{n,l}(r) \simeq \exp(-r/(2\mathrm{L})) \sum_{j=0}^{N-1} h_j L_j(r/\mathrm{L})$$

where $\mathrm{L} > 0$ is the scaling parameter and N is the number of terms in the approximation. The Laguerre pseudospectral method uses as collocation points the roots of the Laguerre polynomial L_N, and gives rise to a generalized matrix eigenproblem $A\,R_{n,l} = E_{n,l}\,B\,R_{n,l}$ where the values of the matrices A and B are given by substituting the approximation on the equation and evaluating it at the collocation points. For more details of the pseudospectral method see [4, 10].

A basic `Matlab` program to compute the radial wave functions $R_{n,l}(r)$ and the allowed energies $E_{n,l}$ for the hydrogen atom with $l = 0$ is shown below. Note that in the Laguerre pseudospectral method we use the ORTHOPOLY library, allowing us a very simple program as all the technicalities related with the orthogonal basis evaluations (and its derivatives) are done with simple calls to the `LagueVal`, `LagueDer`, `LagueDerK` functions of ORTHOPOLY.

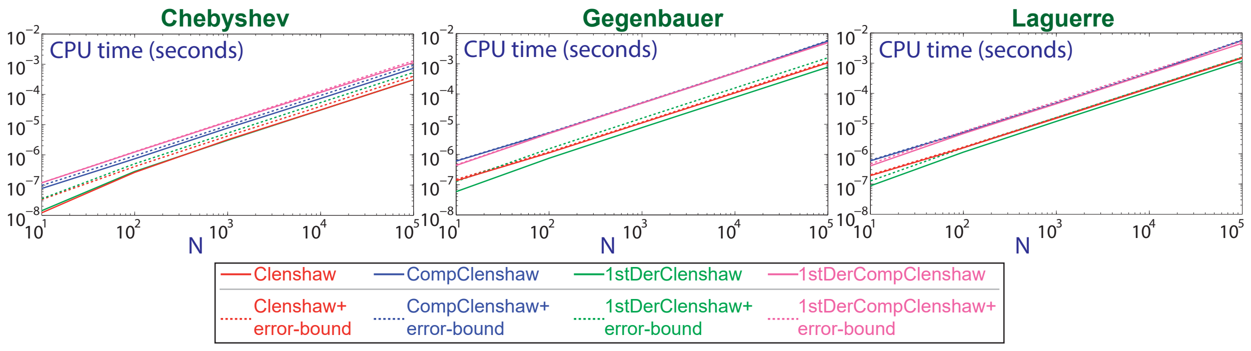

Figure 9: CPU time in the evaluation of polynomial series and their derivatives (with and without error bounds) in Chebyshev, Gegenbauer and Laguerre form using the subroutines `Cheb1Val`, `Cheb1ValwErr`, `CompCheb1Val`, `CompCheb1ValwErr`, `Cheb1Der`, `Cheb1DerwErr`, `CompCheb1Der`, `CompCheb1DerwErr`, `GegenVal`, `GegenValwErr`, `CompGegenVal`, `CompGegenValwErr`, `GegenDer`, `GegenDerwErr`, `CompGegenDer`, `CompGegenDerwErr`, `LagueVal`, `LagueValwErr`, `CompLagueVal`, `CompLagueValwErr`, `LagueDer`, `LagueDerwErr`, `CompLagueDer` and `CompLagueDerwErr` of the C version of the ORTHOPOLY library.

```
N=50; L=9; n=22; % DATA OF THE EXAMPLE
startOrthoPoly % Initialization of the ORTHOPOLY library
    % It has to be on the same folder or to be initialized before.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
J=diag([1:2:2*N-1])-diag([1:N-1],1)-diag([1:N-1],-1);
rho=sort(eig(sparse(J))); r=L*rho; % selected points for
                             %  collocation pseudospectral method
A=zeros(N); B=A;
for i=1:N
    epr=exp(-r(i)/(2*L));
    for j=1:N
      p=zeros(1,j); p(1,j)=1;
      B(i,j)=epr*LagueVal(p,r(i)/L,0);
      A(i,j)=-(B(i,j)/(4*L^2)-epr*(LagueDer(p,r(i)/L,0)-...
             LagueDerK(p,r(i)/L,0,2))/(L^2))/2-B(i,j)/r(i);
    end
end
[hm,E]=eig(A,B);
% hm=matrix of eigenvectors->coefficients of the eigenfunction
% E=energy eigenvalues
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
h=hm(:,n);
x=0:0.01:100; % Radial points of evaluation
u=x;
for i=1:length(x)
% evaluation of the series of the approximate radial wave function
    u(i)=exp(-x(i)/(2*L))*LagueVal(h,x(i)/L,0);
end
```

The program is based on the method explained in [4]. In Figure 10 we show, in a similar way as in [4], the relative errors in the calculations using the Laguerre pseudospectral method (for different values of the scaling parameter L) of the energy eigenvalues $E_{n,0}$ and radial wave functions $R_{n,0}(r)$ for the hydrogen atom with $l = 0$, compared with the exact solution (known for this particular example [5, 41, 42]). We observe that this simple program is able to obtain a quite precise solution in terms of Laguerre orthogonal polyomials, and that depending on the scaling parameter L there is a range of n modes where the obtained energy eigenvalues $E_{n,0}$ present a high accuracy (see left plot on Fig. 10), whereas for small and large n the accuracy is poor. This behaviour is the same observed in [4], and so a combination of several values of the scaling parameter L is recommended for an accurate global solution. Absolute errors for the radial wave functions $R_{n,0}(r)$ approximations for $r \in [0, 100]$ are shown in the right plot of Fig. 10. The approximate solution is given by $R_{n,0}(r) \simeq \exp(-r/(2\mathrm{L})) \sum_{j=0}^{N-1} h_j L_j(r/\mathrm{L})$, where the coefficients of the series development are given by the eigenvectors of the generalized eigenvalue problem, and in the program the evaluation at several values of r (the points `x(i)`) is given by (it uses the `LagueVal` function)

$$R_{n,0}(\texttt{x(i)}) \simeq \texttt{exp(-x(i)/(2*L))*LagueVal(h,x(i)/L,0)}.$$

From the figures we observe that an error $< 10^{-10}$ is obtained in all the cases.

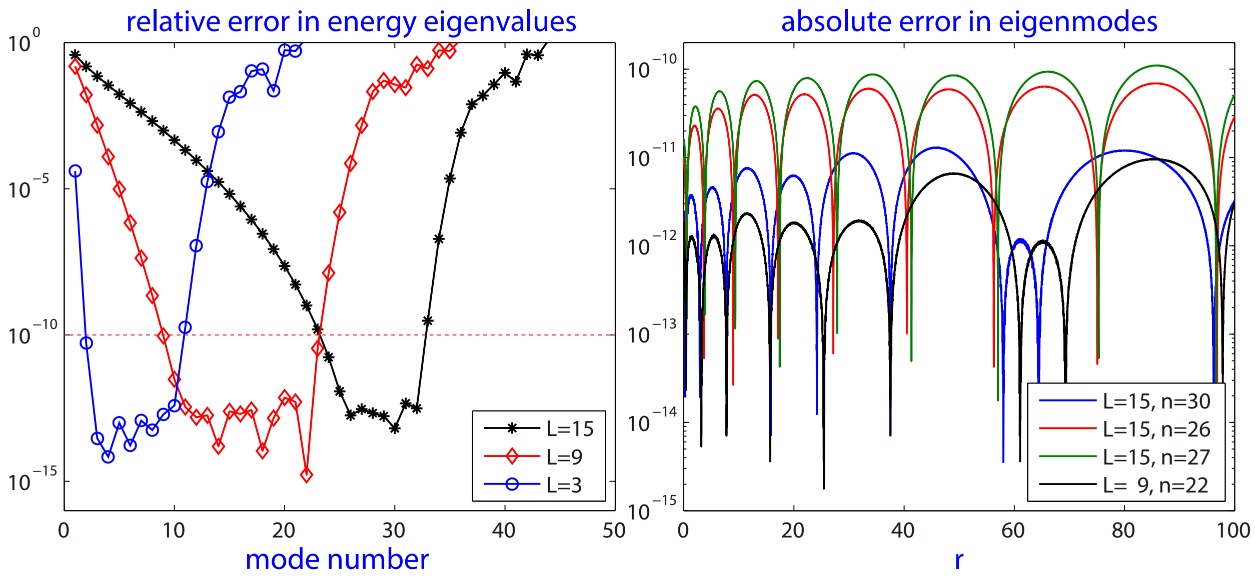

Figure 10: Relative errors in the calculation, using the Laguerre pseudospectral method (for different values of the scaling parameter L), of the energy eigenvalues $E_{n,0}$ (left) and absolute errors in the calculation of the radial wave functions $R_{n,0}(r)$ (right) for the hydrogen atom with $l = 0$.

This example is just a simple example of a direct use of the ORTHOPOLY library in a physical problem. Note that ORTHOPOLY is designed to be used as auxiliary functions for larger programs developed to solve physical, mathematical, chemistry or engineering problems, as it provides accurate algorithms for the evaluation of orthogonal polynomials and polynomial series and their derivatives.

7. Conclusions

This paper presents a practical and highly accurate library (ORTHOPOLY) that provides subroutines for evaluating orthogonal polynomial series in Chebyshev, Legendre, Gegenbauer, Jacobi, Laguerre and Hermite polynomial basis and their derivatives. We use the BCS-algorithm (Barrio-Clenshaw-Smith derivative algorithm) that can scope all these problems at once. The library may also provide running-error bounds of the evaluation processes, and so, it may give information of the accuracy of the results. In cases when high precision is required, a compensated BCS-algorithm is developed that allows to relegate the influence of the conditioning of the problem up to second order in the round-off unit of the computer. The ORTHOPOLY library includes C and `Matlab` versions of all the algorithms. The numerical tests show the accuracy of the algorithms and a simple example on the Schrödinger equation for the radial hydrogen atom illustrates its use in a physical problem.

Acknowledgements

The authors R. B. and S. S. were supported by the Spanish Research project MTM2015-64095-P and by the European Social Fund and Diputación General de Aragón (Grant E48), P. D. by the National Natural Science Foundation of China (No. 61571008) and H. J. by the National Natural Science Foundation of China (No. 61402495, No. 61602166, No. 61303189, No. 61402496). The authors are grateful to the referees for their constructive input.

Bibliography

[1] NIST Digital Library of Mathematical Functions, http://dlmf.nist.gov/, 2016, F. W. J. Olver, A. B. Olde Daalhuis, D. W. Lozier, B. I. Schneider, R. F. Boisvert, C. W. Clark, B. R. Miller and B. V. Saunders, eds.

[2] A. Nakamura, Journal of the Physical Society of Japan **65** (1996) 1589.

[3] A. F. Nikiforov, V. B. Uvarov, *Special Functions of Mathematical Physics: A Unified Introduction with Applications*, Birkhäuser Verlag, Basel., 1988.

[4] J. Boyd, C. Rangan, P. Bucksbaum, Journal of Computational Physics **188** (2003) 56.

[5] H. Alici, H. Taşeli, Applied Numerical Mathematics **87** (2015) 87.

[6] C. W. Clenshaw, Math. Tab. Wash. **9** (1955) 118.

[7] F. J. Smith, Math. Comp. **19** (1965) 33.

[8] D. Funaro, Fortran routines for the spectral methods, Pubblicazioni 891, Instituto di Analisi Numerica del Consiglio Nazionale delle Ricerche, Pavia, Italy, 1993.

[9] V. Gadjokov, J. Jordanova, Computer Physics Communications **31** (1984) 53.

[10] J. Boyd, *Chebyshev and Fourier Spectral Methods*, Dover Publications; Second Edition, 2001.

[11] C. Canuto, M. Y. Hussaini, A. Quarteroni, T. Zang, *Spectral Methods in Fluid Dynamics*, Springer-Verlag, New York, 1988.

[12] D. Gottlieb, S. A. Orszag, *Numerical Analysis of Spectral Methods*, SIAM, Philadelphia, 1977.

[13] R. Baltensperger, J.-P. Berrut, Computers and Mathematics with Applications **37** (1999) 41.

[14] K. Breuer, R. Everson, Journal of Computational Physics **99** (1992) 56.

[15] B. Costa, W. Don, Applied Numerical Mathematics **33** (2000) 151.

[16] W. S. Don, A. Solomonoff, SIAM Journal on Scientific Computing **16** (1995) 1253.

[17] W. Don, A. Solomonoff, SIAM Journal on Scientific Computing **18** (1997) 1040.

[18] Z.-C. Li, S.-Y. Chen, C.-S. Chien, H.-S. Chen, Computer Physics Communications **182** (2011) 1215 .

[19] C.-C. Huang, Computer Physics Communications **180** (2009) 375.

[20] K. Parand, M. Dehghan, A. Rezaei, S. Ghaderi, Computer Physics Communications **181** (2010) 1096.

[21] D. Funaro, *Polynomial Approximation of Differential Equations*, Lecture Notes in Physics, New Series m: Monographs 8, Springer–Verlag, 1992.

[22] T. Tang, M. Trummer, SIAM Journal on Scientific Computing **17** (1996) 430.

[23] A. Gil, J. Segura, N. M. Temme, Computer Physics Communications **210** (2017) 124.

[24] R. Barrio, *Polinomios de Chebyshev; algoritmos y aplicación en la determinación y compresión de órbitas*, PhD thesis, University of Zaragoza, Spain, 1997.

[25] R. Barrio, J. Peña, Applied Numerical Mathematics **43** (2002) 335 .

[26] S. M. Rump, Acta Numerica **19** (2010) 287.

[27] C. Bernardi, Y. Maday, Computers and Structures **30** (1988) 205.

[28] A. Karageorghis, T. Phillips, Journal of Computational Physics **80** (1989) 314.

[29] A. Malek, T. Phillips, IMA Journal of Numerical Analysis **15** (1995) 523.

[30] D. Bailey, R. Barrio, J. Borwein, Applied Mathematics and Computation **218** (2012) 10106.

[31] N. J. Higham, *Accuracy and stability of numerical algorithms*, SIAM, Philadelphia, 1996.

[32] R. Barrio, H. Jiang, S. Serrano, SIAM Journal on Numerical Analysis **51** (2013) 1280.

[33] T. Ogita, S. M. Rump, S. Oishi, SIAM J.Sci. Comput. **26** (2005) 1955.

[34] D. E. Knuth, *The Art of Computer Programming: Seminumerical Algorithms*, Addison-Wesley, third edition, 1998.

[35] T. J. Dekker, Numer. Math. **18** (1971) 224.

[36] H. Jiang, R. Barrio, H. S. Li, X. K. Liao, L. Z. Cheng, F. Su, Appl. Math. Comput. **217** (2011) 9702.

[37] D. H. Bailey, QD library in High-Precision Software Directory, http://crd-legacy.lbl.gov/ dhbailey/mpdist/.

[38] X. S. Li, J. W. Demmel, D. H. Bailey, G. Henry, Y. Hida, J. Iskandar, W. Kahan, S. Y. Kang, A. Kapur, M. C. Martin, B. J. Thompson, T. Tung, D. J. Yoo, ACM Trans. Math. Software. **28** (2002) 152.

[39] R. Barrio, J. M. Peña, Comptes Rendus Mathematique **339** (2004) 293.

[40] R. Barrio, Journal of Computational and Applied Mathematics **138** (2002) 185.

[41] G. W. F. Drake, editor, *Springer Handbook of Atomic, Molecular, and Optical Physics*, Springer, 2006.

[42] D. J. Griffiths, *Introduction to Quantum Mechanics, 2nd Ed.*, Pearson Education, 2005.

AppendixA. C example file: `test_examples_chebyshev.c`

The file `test_examples_chebyshev.c` will produce as result the evaluation of the polynomial $p(x) = (x - 0.75)^7(x-1)^{10}$ in Chebyshev polynomial (of the first kind) series, and the evaluation of the first and fourth derivative, all at $x = 0.65$, using the standard Clenshaw or BCS-algorithm and the compensated versions (with and without running-error bound). The complete list of the coefficients of the test problem appears in the library.

```
#include<stdio.h>
#include<assert.h>
#include<stdlib.h>
#include<math.h>

#include"inline.h"
#include"chebyshev_series.h"

int main ( )
{
  double res;
  double * runerrbound = (double *) malloc(sizeof(double));
  dd_real res_c;
  double x=0.65;
  int n=17;
  double p[]={-28471492267.0/4194304.0,26980769367.0/2097152.0,.....,1.0/65536.0};

  res=Cheb1Val(p,n,x);
  printf("the result of Cheb1Val is %14.14e\n",res);
  res=CompCheb1Val(p,n,x);
  printf("the result of CompCheb1Val is %14.14e\n",res);
  res_c=AccCheb1Val(p,n,x);
  printf("the result of AccCheb1Val is (%14.14e,%14.14e)\n",res_c.H, res_c.L);
  res=Cheb1ValwErr(p,n,x,runerrbound);
  printf("the result of Cheb1ValwErr is %14.14e and the running-error
          bound %e\n",res, *runerrbound);
  res=CompCheb1ValwErr(p,n,x,runerrbound);
  printf("the result of CompCheb1ValwErr is %14.14e and the running-error
          bound %e\n",res, *runerrbound);
  res_c=AccCheb1ValwErr(p,n,x,runerrbound);
  printf("the result of AccCheb1ValwErr is (%14.14e,%14.14e) and the
          running-error bound %e\n",res_c.H,res_c.L, *runerrbound);

  res=Cheb1Der(p,n,x);
  printf("the result of Cheb1Der is %14.14e\n",res);
  res=CompCheb1Der(p,n,x);
  printf("the result of CompCheb1Der is %14.14e\n",res);
  res_c=AccCheb1Der(p,n,x);
  printf("the result of AccCheb1Der is (%14.14e,%14.14e)\n",res_c.H, res_c.L);
  res=Cheb1DerwErr(p,n,x,runerrbound);
  printf("the result of Cheb1DerwErr is %14.14e and the running-error
          bound %e\n",res, *runerrbound);
  res=CompCheb1DerwErr(p,n,x,runerrbound);
  printf("the result of CompCheb1DerwErr is %14.14e and the running-error
          bound %e\n",res, *runerrbound);
  res_c=AccCheb1DerwErr(p,n,x,runerrbound);
  printf("the result of AccCheb1DerwErr is (%14.14e,%14.14e) and the
          running-error bound %e\n",res_c.H,res_c.L, *runerrbound);
  res=Cheb1DerK(p,n,x,4);
  printf("the result of Cheb1DerK is %14.14e\n",res);
  res=CompCheb1DerK(p,n,x,4);
  printf("the result of CompCheb1DerK is %14.14e\n",res);
  res_c=AccCheb1DerK(p,n,x,4);
  printf("the result of AccCheb1DerK is (%14.14e,%14.14e)\n",res_c.H, res_c.L);
  res=Cheb1DerKwErr(p,n,x,runerrbound,4);
  printf("the result of Cheb1DerKwErr is %14.14e and the running-error
          bound %e\n",res, *runerrbound);
  res=CompCheb1DerKwErr(p,n,x,runerrbound,4);
  printf("the result of CompCheb1DerKwErr is %14.14e and the running-error
          bound %e\n",res, *runerrbound);
  res_c=AccCheb1DerKwErr(p,n,x,runerrbound,4);
  printf("the result of AccCheb1DerKwErr is (%14.14e,%14.14e) and the
          running-error bound %e\n",res_c.H,res_c.L, *runerrbound);

  return 0;
}
```

The output values of the test example are:

```
the result of Cheb1Val is -2.72848410531878e-12
the result of CompCheb1Val is -2.75854735351562e-12
the result of AccCheb1Val is (-2.75854735351562e-12,-1.00974195868290e-28)
the result of Cheb1ValwErr is -2.72848410531878e-12
    and the running-error bound 6.780968e-11
the result of CompCheb1ValwErr is -2.75854735351562e-12
    and the running-error bound 6.093031e-27
the result of AccCheb1ValwErr is (-2.75854735351562e-12,-1.00974195868290e-28)
    and the running-error bound 5.992057e-27
the result of Cheb1Der is 2.76486389338970e-10
the result of CompCheb1Der is 2.71913953417968e-10
the result of AccCheb1Der is (2.71913953417968e-10,-4.03896783473158e-27)
the result of Cheb1DerwErr is 2.76486389338970e-10
    and the running-error bound 4.480965e-10
the result of CompCheb1DerwErr is 2.71913953417968e-10
    and the running-error bound 2.684536e-26
the result of AccCheb1DerwErr is (2.71913953417968e-10,-4.03896783473158e-27)
    and the running-error bound 2.280639e-26
the result of Cheb1DerK is -1.54347508214414e-04
the result of CompCheb1DerK is -1.54346756238281e-04
the result of AccCheb1DerK is (-1.54346756238281e-04,-5.11280336619026e-21)
the result of Cheb1DerKwErr is -1.54347508214414e-04
    and the running-error bound 8.853229e-08
the result of CompCheb1DerKwErr is -1.54346756238281e-04
    and the running-error bound 5.120871e-21
the result of AccCheb1DerKwErr is (-1.54346756238281e-04,-5.11280336619026e-21)
    and the running-error bound 8.067393e-24
```

AppendixB. `Matlab` example file: `test_examples_chebyshev.m`

The file `test_examples_chebyshev.m` will produce as result the evaluation of the polynomial $p(x) = (x - 0.75)^7(x-1)^{10}$ in Chebyshev polynomial (of the first kind) series, and the evaluation of the first and fourth derivative, all at $x = 0.65$, using the standard Clenshaw or BCS-algorithm and the compensated versions (with and without running-error bound). The complete list of the coefficients of the test problem appears in the library.

```
p=[-28471492267/4194304,26980769367/2097152,...,1/65536];
x=0.65;

 % Cheb1Val evaluates a series of Chebyshev polynomial at the point x, which is in [-1, 1]
val=Cheb1Val(p,x)
 % CompCheb1Val evaluates a series of Chebyshev polynomial  at the point x,
 % which is in [-1, 1], with compensated method
cval=CompCheb1Val(p,x)
 % Cheb1ValwErr evaluates a series of Chebyshev polynomial at the point x,
 % which is in [-1, 1], and performs a running-error bound
[val2,err]=Cheb1ValwErr(p,x)
 % CompCheb1ValwErr evaluates a series of Chebyshev polynomial  at the point x,
 % which is in [-1, 1], with compensated method, and performs a running-error bound
[cval2,cerr]=CompCheb1ValwErr(p,x)
 % Cheb1Der evaluates the first derivative of a series of Chebyshev polynomial
 % at the point x, which is in [-1, 1]
der=Cheb1Der(p,x)
 % CompCheb1Der evaluates the first derivative of a series of Chebyshev polynomial
 % at the point x, which is in [-1, 1], with compensated method
cder=CompCheb1Der(p,x)
 % Cheb1DerwErr evaluates  the first derivative of a series of Chebyshev polynomial
 % at the point x, which is in [-1, 1], and performs a running-error bound
[der2,derr]=Cheb1DerwErr(p,x)
 % CompCheb1DerwErr evaluates  the first derivative of a series of Chebyshev polynomial
 % at the point x, which is in [-1, 1], with compensated method, and performs
 % a running-error bound
[cder2,cderr]=CompCheb1DerwErr(p,x)
 % Cheb1DerK evaluates  the k-th derivative of a series of Chebyshev polynomial
 % at the point x, which is in [-1, 1], in this case k=4
sder=Cheb1DerK(p,x,4)
 % CompCheb1DerK evaluates  the k-th derivative of a series of Chebyshev polynomial
 % at the point x, which is in [-1, 1], with compensated method, in this case k=4
csder=CompCheb1DerK(p,x,4)
 % Cheb1DerKwErr evaluates  the k-th derivative of a series of Chebyshev polynomial
 % at the point x, which is in [-1, 1], and performs a running-error bound, in this case k=4
[sder2,sderr]=Cheb1DerKwErr(p,x,4)
 % CompGegenDerKwErr evaluates  the k-th derivative of a series of Gegenbauer polynomial
 % at the point x, which is in [-1, 1], with compensated method, and performs
 % a running-error bound, in this case k=4
[csder2,csderr]=CompCheb1DerKwErr(p,x,4)
```

The output values of the test example are:

```
val    = -2.728484105318785e-012
cval   = -2.758547353515619e-012
val2   = -2.728484105318785e-012
err    =  6.780967690622450e-011
cval2  = -2.758547353515619e-012
cerr   =  6.093030937654751e-027
der    =  2.719389158301055e-010
```

```
cder   =  2.719139534179682e-010
der2   =  2.719389158301055e-010
derr   =  4.480965185261817e-010
cder2  =  2.719139534179682e-010
cderr  =  2.684535685107712e-026
sder   = -1.543461985420436e-004
csder  = -1.543467562382810e-004
sder2  = -1.543461985420436e-004
sderr  =  8.853229043816396e-008
csder2 = -1.543467562382810e-004
csderr =  5.120870759223418e-021
```