We describe a procedure for the determination of the roots of functions satisfying second-order ordinary differential equations, including the classical special functions. The scheme is based on a combination of the Prüfer transform with the classical Taylor series method for the solution of ordinary differential equations, and requires $O(1)$ operations for the determination of each root. When the functions in question are classical orthogonal polynomials (Legendre, Hermite, etc.), the techniques presented here also provide tools for the evaluation of the weights for the associated Gaussian quadratures. The performance of the scheme for several classical special functions (prolate spheroidal wave functions, Bessel functions, and Legendre, Hermite, and Laguerre polynomials) is illustrated with numerical examples.

A Fast Algorithm for the Calculation of the Roots of Special Functions

Andreas Glaser, Xiangtao Liu, and Vladimir Rokhlin
Technical Report YALEU/DCS/TR-1367
August 23, 2006

The authors were supported in part by the U.S. Department of Defense under DARPA Grant #FA9550-06-1-0239 and AFOSR Grant #FA9550-06-1-0197.
Approved for public release: Distribution is unlimited.
Keywords: *Roots, Special Functions, Gaussian Quadratures, Sturm-Liouville*

1 Introduction

Zeroes of classical special functions play an important role in the mathematical sciences, being related to Gaussian quadratures (both classical and generalized), resonances in mechanical and electrical systems, quantum mechanical calculations, etc. As a result, algorithms for the determination of roots of functions of this type are quite well developed; most of them are based on the existence of three-term recurrence relations for the classical special functions, and the associated connection to the eigenvalues of certain tridiagonal matrices (see, for example, [6, 8, 2]). The latter are dealt with via well-understood and reliable techniques, such as the QR iteration.

While such methods are quite efficient for small-scale problems, they require order n^2 operations, where n is the the number of roots to be computed. Relatively recently, algorithms for the diagonalization of tridiagonal matrices were introduced with CPU time requirements proportional to $n \cdot \log(n)$ (see [7]); a detailed description of the resulting fast root finding scheme is given in [12]. In this paper we present a scheme which is unrelated to the three term recurrence relations and tends to be significantly faster for large n.

The subject of this paper is the fact that the roots of classical special functions can be found for a cost proportional to n, using the fact that such functions satisfy ordinary differential equations of the form

$$p(x)\,\frac{d^2u}{dx^2}(x) + q(x)\,\frac{du}{dx}(x) + r(x)\,u(x) = 0, \tag{1}$$

where $p(x)$, $q(x)$ and $r(x)$ are polynomials of degree 2. Using classical tools such as Taylor series and the Prüfer transformation, our approach leads to remarkably simple and efficient schemes for the construction of classical Gaussian quadratures, roots of Bessel and prolate spheroidal wave functions, and the roots of other functions satisfying a second-order differential equation with polynomial coefficients.

2 Mathematical and numerical preliminaries

In this section we summarize several well-known facts to be used in this paper; all of these can be found in [1, 3, 4, 11, 5]. Throughout this paper, the derivative of a function $f : \mathbb{R} \rightarrow \mathbb{R}$ will be denoted by f' and p, q, r will denote the second-order polynomials introduced in equation (1). All functions are assumed to be sufficiently smooth.

2.1 The Prüfer transform

Given equation (1) and a differentiable positive function $\gamma : \mathbb{R} \to \mathbb{R}$, we define the function $\theta : \mathbb{R} \to \mathbb{R}$ as the solution of the differential equation

$$\theta' = -\frac{\gamma}{p}\sin^2(\theta) - \frac{r}{\gamma}\cos^2(\theta) - \left(\frac{\gamma'}{\gamma} + \frac{q-p'}{p}\right)\frac{\sin(2\theta)}{2}, \tag{2}$$

where θ, γ, p, q, r are functions of x; integrating both sides of (2) with respect to x and carrying out elementary manipulations, we obtain

$$\theta(x) = \arctan\left(\frac{1}{\gamma(x)}\frac{p(x)u'(x)}{u(x)}\right), \tag{3}$$

where u is the function defined by equation (1); we observe that, for any $\tilde{x}$ such that $u(\tilde{x}) = 0$,

$$\theta(\tilde{x}) = (n + 1/2)\pi, \tag{4}$$

with n an integer. Similarly, for any $\tilde{x}$ such that $u'(\tilde{x})$=0,

$$\theta(\tilde{x}) = n\pi, \tag{5}$$

where again n is an integer.

Observation 1 It often happens that the function θ is better behaved (both numerically and analytically) than the function u that gave rise to it. In such cases, it becomes attractive to solve equations (4) and (5) instead of equations

$$u(x) = 0, \tag{6}$$

$$u'(x) = 0, \tag{7}$$

respectively. This observation is not new, and in fact the above construction is a variant of the classical Prüfer transform, which can be found, for example, in [3].

The choice of the function γ in (3) influences the behavior of the function θ, without changing the solutions of equations (4), (5) (the latter are also the solutions of equations (6), (7), respectively, with u being independent of γ). For example, if $\gamma \equiv 1$ and equation (1) is self-adjoint (i.e., $q = p'$), then

$$\theta' = -\frac{1}{p}\sin^2(\theta) - r\cos^2(\theta) \tag{8}$$

and the function θ is strictly decreasing if p and r are everywhere positive.

However, it often happens that $1/p$ and r are of different orders of magnitude; this causes the value of θ' to vary dramatically on the interval $\theta = [-\pi/2, \pi/2]$, making it inconvenient when used as a numerical tool. Figure 1(a) illustrates the

last point. It shows θ for a Legendre polynomial of order $l = 100$, with $p = 1 - x^2$, $q = p'$ and $r = l(l+1)$, in the interval between the polynomial's greatest negative root and its least positive root.

Such step-like behavior complicates many numerical operations involving (2), and can be controlled via an appropriate choice of γ. Indeed, choosing

$$\gamma \equiv \sqrt{rp} \tag{9}$$

converts (3) into

$$\theta(x) = \arctan\left(\frac{1}{\sqrt{r(x)p(x)}}\frac{p(x)u'(x)}{u(x)}\right), \tag{10}$$

and substituting (9) into (2) yields

$$\theta' = -\sqrt{\frac{r}{p}} - \frac{r'p - p'r + 2rq}{2rp} \cdot \frac{\sin(2\theta)}{2}. \tag{11}$$

In this case, the large values of the product rp do not cause excessive variations in the magnitude of θ', which is illustrated by Figure 1(b), which shows θ for the same Legendre polynomial as in Figure 1(a), but with γ defined in (9).

Whenever $\frac{d\theta}{dx} < 0$, the ODE (11) can be rewritten in the form

$$\frac{dx}{d\theta} = -\left(\sqrt{\frac{r}{p}} + \frac{r'p - p'r + 2rq}{2rp} \cdot \frac{\sin(2\theta)}{2}\right)^{-1}, \tag{12}$$

where θ, γ, r, q, p are functions of x, and equation (12) can be viewed as a differential equation defining x as a function of θ. Clearly, θ' is guaranteed to be negative whenever

$$\left|\frac{r'p - p'r + 2rq}{4rp}\right| < \sqrt{\frac{r}{p}}, \tag{13}$$

and the latter condition is easy to test for, and is satisfied by all standard special functions.

2.2 Taylor series for special functions

As is well known, any function $u : \mathbb{R} \to \mathbb{R}$, that is sufficiently smooth in the neighborhood of the point $x_0 \in \mathbb{R}$ can be approximated by its Taylor series

$$u(x_0 + h) = \sum_{k=0}^{m} \frac{u^{(k)}(x_0)}{k!} h^k + \varepsilon, \tag{14}$$

with

$$|\varepsilon| \leq \sup_{|x - x_0| \leq h} \left\{\frac{u^{m+1}(x)}{(m+1)!} h^{m+1}\right\}, \tag{15}$$

where $u^{(k)}$ denotes the k^{th} derivative of u; the Taylor series for u' is

$$u'(x_0 + h) = \sum_{k=1}^{m} \frac{u^{(k)}(x_0)}{k-1!} h^{k-1} + \tilde{\varepsilon}, \tag{16}$$

with

$$|\tilde{\varepsilon}| \leq \sup_{|x-x_0| \leq h} \left\{ \frac{u^{m+1}(x)}{m!} h^m \right\}. \tag{17}$$

If u satisfies equation (1), and p, q, r are polynomials of order 2, then the consecutive derivatives of u can be obtained via a simple recursion. Indeed, differentiating (1) k times and carrying out elementary manipulations, we have

$$\begin{aligned} pu^{(k+2)} &= -(kp' + q)\, u^{(k+1)} - \left(\frac{k(k-1)}{2} p'' + kq' + r \right) u^{(k)} \\ &\quad - \left(\frac{k(k-1)}{2} q'' + kr' \right) u^{(k-1)} - \frac{k(k-1)}{2} r'' u^{(k-2)}, \end{aligned} \tag{18}$$

for any $k \geq 0$.

2.3 Numerical tools

In this section we summarize several numerical techniques to be used in the paper. All of them can be found, for example, in [4].

2.3.1 A second-order Runge-Kutta method

The second-order Runge-Kutta method (also known as the Heun or midpoint method) solves the initial-value problem

$$\begin{aligned} y(x_0) &= y_0, \\ y'(x) &= f(x, y), \end{aligned} \tag{19}$$

on the interval $[x_0, x_0 + L]$ by taking a sequence of n steps:

$$\begin{aligned} x_{i+1} &= x_i + h, \\ k_1 &= h\, f(x_i, y_i), \\ k_2 &= h\, f(x_i + h, y + k_1), \\ y_{i+1} &= y_i + \frac{1}{2}(k_1 + k_2). \end{aligned} \tag{20}$$

where $h = L/n$. The cost of this algorithm is of order n, and the global truncation error is of order h^2.

2.3.2 The Taylor series method for the solution of ODEs

The Taylor series method is a classical numerical scheme for the solution of ODEs. Given equation (1), the values of $u(x_k), u'(x_k)$ at the point x_k, and an appropriately chosen integer $m \geq 2$, the Taylor series method calls for repeated differentiation of (1), resulting in the values of $u'''(x_k), u^{(4)}(x_k), \ldots, u^{(m-1)}(x_k), u^{(m)}(x_k)$. The latter are used to evaluate an approximations to $u(x_{k+1}), u'(x_{k+1})$ via the expansions (14), (16), with $h = x_{k+1} - x_k$. Obviously, the truncation error of one step of the Taylor series method is h^m, and its cost is determined by the cost of finding the derivatives $u'''(x_k), \ldots, u^{(m-1)}(x_k), u^{(m)}(x_k)$; in the case of (1), the recursion (18) permits $u'''(x_k), \ldots, u^{(m-1)}(x_k), u^{(m)}(x_k)$ to be evaluated in $O(m)$ operations.

By choosing the order m of the expansions (14), (16), one can control the convergence rate of the Taylor series method, and often it makes sense to choose very high orders, obtaining essentially machine precision. In general, Taylor series algorithms are not immune to stability problems; however, due to the following observation this issue does not arise in the algorithm of this paper.

Observation 2 Somewhat surprisingly, stability issues are obviated when all of the points $x_1, x_2, \ldots$ are roots of u or u'. Indeed, given the recursion formula (18), the expressions (14), (16) can be viewed as a linear mapping $M : \mathbb{R}^2 \rightarrow \mathbb{R}^2$ converting the pair $u(x_k), u'(x_k)$ into the pair $u(x_{k+1}), u'(x_{k+1})$; the stability of the resulting ODE solver is determined by the eigenvalues of M. However, if

$$u(x_k) = 0, \tag{21}$$

$$u(x_{k+1}) = 0, \tag{22}$$

then the mapping $M : \mathbb{R}^2 \rightarrow \mathbb{R}^2$ is replaced with a mapping $\widetilde{M} : \mathbb{R}^1 \rightarrow \mathbb{R}^1$, converting $u'(x_k)$ into $u'(x_{k+1})$. In other words, the 2-dimensional mapping M has been replaced with a simple scaling, not affecting the roots of either u or of u'. An analogous argument applies when equation (21) is replaced by

$$u'(x_k) = 0, \tag{23}$$

or equation (22) is replaced by

$$u'(x_{k+1}) = 0, \tag{24}$$

or when both equations are replaced.

2.3.3 Newton's method

Given an initial approximation x_0, Newton's method solves the equation $f(x) = 0$ iteratively, with the n^{th} iteration defined by the formula

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}. \tag{25}$$

If $\tilde{x}$ is a simple root of f and x_0 has been chosen sufficiently close to $\tilde{x}$, then the convergence is quadratic, i.e., the number of correct digits of the approximation x_n will double after each iteration.

2.4 Orthogonal polynomials, Bessel functions, and prolate spheroidal wave functions

This section summarizes certain well-known properties of several classes of special functions, used in Section 4 to test the performance of the algorithm of this paper; all of the facts in this section can be found in [1, 11, 5].

2.4.1 Orthogonal polynomials

Given a (possibly infinite) interval (a, b) and a positive function w, such that the product of w with any polynomial is integrable on (a, b), we say that $p_0, p_1, \ldots, p_n$ is an orthogonal family of polynomials on (a, b) with weight w, if for each $k = 1, 2, 3, \ldots$, the polynomial p_k is of degree k and

$$\int_a^b p_i(x)\, p_j(x)\, w(x)\, dx = \begin{cases} 0, & i \neq j \\ a_i, & i = j \end{cases}, \tag{26}$$

where $a_i > 0$.

In this subsection we briefly discuss properties of Legendre, Hermite, and Laguerre polynomials, which are classical examples of orthogonal families of polynomials. Each of these classical families satisfies a homogenous second-order ordinary differential equation.

The Legendre polynomials $P_0, P_1, \ldots, P_n$ are the orthogonal family with respect to the weight function $w(x) = 1$ on the interval $(-1, 1)$ for $k = 1, \ldots, n$. The n^{th} degree polynomial P_n of this family satisfies the differential equation

$$(1 - x^2)P_n''(x) - 2x\, P_n'(x) + n(n+1)P_n(x) = 0. \tag{27}$$

The Hermite polynomials $H_1, H_2, \ldots, H_n$ are the orthogonal family with respect to the weight function $w(x) = \exp(-x^2)$ on the interval $(-\infty, \infty)$, and the n^{th} degree Hermite polynomial H_n satisfies the equation

$$H_n''(x) - 2x\, H_n'(x) + 2n\, H_n(x) = 0. \tag{28}$$

Finally, the Laguerre polynomials $L_1, L_2, \ldots, L_n$ are orthogonal with respect to the weight function $w(x) = \exp(-x)$ on the interval $(0, \infty)$, and the n^{th} degree Laguerre polynomial L_n satisfies the equation

$$x\, L_n''(x) + (1 - x)L_n'(x) + n\, L_n(x) = 0. \tag{29}$$

Every class of orthogonal polynomials satisfies a three-term recurrence relation, i.e., there exist real sequences $c_0, c_1, c_2, c_3, \ldots$, $d_0, d_1, d_2, d_3, \ldots$, such that, for any $m \geq 1$,

$$c_m\, p_{m+1}(x) = (x - d_m)\, p_m(x) - c_{m-1}\, p_{m-1}(x). \tag{30}$$

Thus, given $x \in \mathbb{R}$, an integer $n > 2$, and the values $p_0(x), p_1(x)$, it follows from (30) that the values $p_2(x), p_3(x), p_4(x), \ldots, p_n(x)$ can be evaluated in $O(n)$ operations.

Differentiating (30), we obtain the recurrence

$$c_m\, p'_{m+1}(x) = (x - d_m)\, p'_m(x) - c_{m-1}\, p'_{m-1}(x) + p_m(x), \tag{31}$$

which can be used for the efficient evaluation of the derivatives of $p_2, p_3, \ldots$.

Specifically, for the Legendre polynomials the recurrence relations for P_n and P'_n are (see 22.7.10, [1])

$$P_{n+1}(x) = \frac{2n+1}{n+1} x\, P_n(x) - \frac{n}{n+1} P_{n-1}(x), \tag{32}$$

$$P'_{n+1}(x) = \frac{2n+1}{n+1} \left(x\, P'_n(x) + P_n(x)\right) - \frac{n}{n+1} P'_{n-1}(x), \tag{33}$$

with the initial values $P_{-1}(x) = 0$, $P_0(x) = 1$, $P'_{-1}(x) = 0$, $P'_0(x) = 0$; for the Hermite polynomials the recurrence relations for H_n and H'_n are (see 22.7.13, [1])

$$H_{n+1}(x) = 2x\, H_n(x) - 2n\, H_{n-1}(x), \tag{34}$$

$$H'_{n+1}(x) = 2\left(x\, H'_n(x) + H_n(x)\right) - 2n\, H'_{n-1}(x), \tag{35}$$

with the initial values $H_{-1}(x) = 0$, $H_0(x) = 1$, $H'_{-1}(x) = 0$, $H'_0(x) = 0$; and for the Laguerre polynomials the recurrence relations for L_n and L'_n are (see 22.7.12, [1])

$$L_{n+1}(x) = \frac{2n+1-x}{n+1} L_n(x) - \frac{n}{n+1} L_{n-1}(x), \tag{36}$$

$$L'_{n+1}(x) = \frac{2n+1-x}{n+1} L'_n(x) - \frac{1}{n+1} L_n(x) - \frac{n}{n+1} L'_{n-1}(x), \tag{37}$$

with the initial values $L_{-1}(x) = 0$, $L_0(x) = 1$, $L'_{-1}(x) = 0$, $L'_0(x) = 0$.

The Legendre and Hermite polynomials of even degrees are even and have a local extremum at $x = 0$; for odd degrees they are odd and have a root at $x = 0$. We will also make use of the fact that the first (smallest) root x_1^n of the n^{th} Laguerre polynomial satisfies

$$x_1^n > \frac{2}{4n+2}. \tag{38}$$

Orthogonal polynomials are a classical tool for the design of numerical integration rules known as Gaussian quadratures. Given the roots $x_1, \ldots, x_n$ of the n^{th} degree

polynomial p_n of an orthogonal family on the interval (a,b) with weight function w, there exist real numbers $w_1^n, \ldots, w_n^n$ such that the formula

$$\int_a^b f(x)\, w(x)\, dx \approx \sum_{k=1}^{n} w_k^n\, f(x_k) \tag{39}$$

is exact for any f that is a polynomial of degree $2n-1$ or less.

The quadrature weights for the Legendre polynomials are (see 15.3.1, [11])

$$w_k^n = \frac{2}{(1-x_k^2)P_n'^2(x_k)}; \tag{40}$$

the Hermite quadrature weights are (see 15.3.6, [11])

$$w_k^n = \frac{\pi^{\frac{1}{2}} 2^{n+1} n!}{H_n'^2(x_k)}, \tag{41}$$

and the quadrature weights for the Laguerre case are (see 15.3.5, [11])

$$w_k^n = \frac{1}{x_k\, L_n'^2(x_k)}. \tag{42}$$

Remark 3 The Legendre, Hermite and Laguerre polynomials satisfy the bounds (see, for example, 22.14, [1])

$$|P_n(x)| \leq 1, \tag{43}$$

$$|H_n(x)| < \pi^{\frac{1}{4}} 2^{n/2} \sqrt{n!}\, e^{x^2/2}, \tag{44}$$

$$|L_n(x)| \leq e^{x/2}. \tag{45}$$

The bounds (44) and (45) indicate that the Hermite and the Laguerre polynomials are likely to cause problems when used as a numerical tool. Even for moderate values of x their values can become so large that it exceeds the maximum value representable in standard floating-point arithmetic.

This overflow problem can easily be avoided by working with the functions

$$\widetilde{H}_n(x) = \frac{e^{-x^2/2}}{\pi^{\frac{1}{4}} 2^{n/2} \sqrt{n!}} H_n(x), \tag{46}$$

$$\widetilde{L}_n(x) = e^{-x/2} L_n(x), \tag{47}$$

called normalized Hermite and Laguerre functions, instead of working with the polynomials directly. The functions $\widetilde{H}_n(x)$ and $\widetilde{L}_n(x)$ have the same roots as $H_n(x)$ and $L_n(x)$, and (44), (45) guarantee that they are bounded by one. Equations (28), (29) imply the following differential equations for $\widetilde{H}_n(x)$ and $\widetilde{L}_n(x)$

$$\widetilde{H}_n''(x) + (2n+1-x^2)\widetilde{H}_n(x) = 0, \tag{48}$$

$$x^2\widetilde{L}_n''(x) + x\,\widetilde{L}_n'(x) - \left(\frac{1}{4}x^2 - \left(n+\frac{1}{2}\right)x\right)\widetilde{L}_n(x) = 0. \tag{49}$$

The functions $\widetilde{H}_n$ can be evaluated via the recurrence

$$\widetilde{H}_{n+1}(x) = \sqrt{\frac{2}{n+1}}x\,\widetilde{H}_n(x) - \sqrt{\frac{n}{n+1}}\widetilde{H}_{n-1}(x), \tag{50}$$

with the initial values $\widetilde{H}_{-1} = 0$, $\widetilde{H}_0 = \pi^{-\frac{1}{4}}e^{-x^2/2}$. Furthermore, if x is a root of H_n, then $\widetilde{H}_n'(x)$ can be evaluated via the recurrence

$$\widetilde{H}_{n+1}'(x) = \sqrt{\frac{2}{n+1}}\left(x\,\widetilde{H}_n'(x) + \widetilde{H}_n(x)\right) - \sqrt{\frac{n}{n+1}}\widetilde{H}_{n-1}'(x), \tag{51}$$

with the initial values $\widetilde{H}_{-1} = 0$, $\widetilde{H}_0 = 0$. Combining (46) with (41), we observe that

$$w_k^n = \frac{2\,e^{-x_k^2}}{\widetilde{H}_n'^2(x_k)}. \tag{52}$$

The functions $\widetilde{L}_n$ can be evaluated via the recurrence

$$\widetilde{L}_{n+1}(x) = \frac{2n+1-x}{n+1}\widetilde{L}_n(x) - \frac{n}{n+1}\widetilde{L}_{n-1}(x), \tag{53}$$

the same recurrence relation as for L_n, with the initial values $\widetilde{L}_{-1}(x) = 0$, $\widetilde{L}_0(x) = e^{-x/2}$. If x is a root of $\widetilde{L}_n$, then $\widetilde{L}_n'(x)$ can be evaluated via the recurrence

$$\widetilde{L}_{n+1}'(x) = \frac{2n+1-x}{n+1}\widetilde{L}_n'(x) - \frac{1}{n+1}\widetilde{L}_n(x) - \frac{n}{n+1}\widetilde{L}_{n-1}'(x), \tag{54}$$

with the initial values $\widetilde{L}_{-1}(x) = 0$, $\widetilde{L}_0(x) = 0$, $\widetilde{L}_{-1}'(x) = 0$, $\widetilde{L}_0'(x) = 0$. Finally, combining (47) and (42), we obtain

$$w_k^n = \frac{e^{-x_k}}{x_k\widetilde{L}_n'^2(x_k)}. \tag{55}$$

2.4.2 Bessel functions

The n^{th} order Bessel function J_n of the first kind satisfies the differential equation

$$x^2J_n''(x) + x\,J_n'(x) + (x^2 - n^2)J_n(x) = 0. \tag{56}$$

Figure 2 shows a plot of J_{50}. The Bessel function of order n starts to oscillate around $x = n$, so that with x_1 denoting the smallest positive root of $J_n(x)$ (see [1], 9.5.14)

$$x_1 > n + n^{\frac{1}{3}}. \tag{57}$$

For large values of x, the asymptotic behavior of J_n is given by (see [1], 9.2.5)

$$J_n(x) \approx \sqrt{\frac{2}{\pi x}} \cos\left(x - \left(n + \frac{1}{2}\right)\frac{\pi}{2}\right) + O\left(\frac{1}{x^{1.5}}\right). \tag{58}$$

Remark 4 There exist various classical schemes for the evaluation of $J_n(x)$ in $O(n)$ operations. See, for example, 9.12, [1] for $J_n(x)$ and 9.1.27, [1] for $J_n'(x)$.

2.4.3 Prolate spheroidal wave functions

The prolate spheroidal wave functions (PSWFs) corresponding to the band-limit c are the eigenvectors of the integral operator G_c defined via the formula

$$F_c(\psi)(x) = \int_{-1}^{1} e^{icxt}\psi(t)\,dt; \tag{59}$$

they are also the eigenvectors of the differential operator

$$G_c(\psi)(x) = (1-x^2)\psi''(x) - 2x\,\psi'(x) - c^2x^2\psi(x). \tag{60}$$

The n^{th} eigenvalue of the operator (60) is normally denoted by χ_n, so that the n^{th} eigenvector ψ_n^c of (60) satisfies the ODE

$$(1-x^2)\psi_n''(x) - 2x\,\psi_n'(x) + (\chi_n^c - c^2x^2)\psi_n(x) = 0, \tag{61}$$

which provides the standard tools for the evaluation of the functions ψ_n, eigenvalues χ_n^c, and related quantities.

The function ψ_n has n roots; for even n the function ψ_n is even and has a local extreme value at $x = 0$; for odd n the function ψ_n is odd and has a root at $x = 0$. We refer the reader to [13, 10] for more detailed information about PSWFs.

3 Algorithm

In this section we describe an algorithm for computing the roots of a function u which is a solution of equation (1). We give an informal outline in the section below, followed by a detailed description in Section 3.2.

3.1 Informal description

The algorithm of this paper finds all roots of a function u satisfying equation (1) recursively by calculating each successively bigger root from the previous one.

3.1.1 First root

Given a starting value x_s, the algorithm calculates the smallest root x_1 that is larger or equal to x_s in two steps. First, it finds an approximation $\tilde{x_1}$ to the initial root by calculating $\theta_0 = \theta(x_s)$ via equation (10) and solving the differential equation (12) with the initial condition $x(\theta_0) = x_s$ in the interval $\theta \in (\theta_0, -\pi/2)$ via the Runge-Kutta method (20). Due to (4), the value of x at $-\pi/2$ is a root of u.

Remark 5 In our experience, ten steps of the Runge-Kutta method are sufficient to yield x_1 with two to three digits of accuracy. Using a higher order scheme for the solution of the differential equation (12) and increasing the number of nodes in the discretization of the interval $(\pi/2, -\pi/2)$, one could obtain the root x_1 with arbitrarily high precision. While this scheme could be continued to find all roots $x_1, x_2, \ldots, x_n$ for a cost proportional to n, its actual cost would be much greater than the cost of the scheme of this paper, where we solve the equation (12) to low precision, and refine the solution via the Newton procedure described below.

The second step uses the Newton procedure (25) to obtain the root x_1, starting with the approximation $\tilde{x}_1$. Since the Newton process is quadratically convergent, it takes about three steps per root to obtain 15-digit accuracy, and each of the steps requires the evaluation of u and u' at a point in the vicinity of x_{i+1}.

The evaluation of u and u' are performed by a scheme specific to the function u. If the evaluation cost is k operations then the cost for calculating the first root is of order k. For orthogonal polynomials, for example, u and u' can be evaluated via the recurrence relations (30), (31) for a cost which is of order of the degree of the polynomial.

Remark 6 In cases like the Legendre or the Hermite polynomials, where an initial root or an extreme value can be obtained via symmetry considerations, the computation of the first root can be simplified substantially. These details are discussed in Sections 3.2, 4 below.

3.1.2 Subsequent roots

Given a root x_i of u, the algorithm finds the next bigger root x_{i+1} of u in two steps. Analogous to the computation of the first root, the first step finds an approximation $\tilde{x}_{i+1}$ by solving the differential equation (12) via the Runge-Kutta scheme (20) on the interval $\theta \in (\pi/2, -\pi/2)$ with the initial condition $x(\pi/2) = x_i$. Due to (4), $\tilde{x}_{i+1}$ is an approximation to the root x_{i+1}.

The second step uses the Newton procedure (25) to increase the accuracy of the approximation $\tilde{x}_{i+1}$. However, now the evaluation of u and u' is performed via expansion (14), where the required derivatives of u are computed via the recurrence (18).

Remark 7 The required number of terms in the Taylor expansion has been determined numerically. For all examples shown in this paper, 30 terms are sufficient for double precision (15-digit), and 60 terms are sufficient for extended precision (30-digit).

The Taylor expansion enables the evaluation of u and u' at a reasonably small constant cost. Adding up the costs for all n roots, we observe that the total cost of the algorithm is of the order $n + k$ operations, where k is the cost for evaluating u and u'.

The repeated evaluation of u via its Taylor expansion can be seen as an instance of the Taylor series method. Since the steps are taken from one root to another, Observation 2 obviates the issue of numerical stability of the procedure.

Remark 8 The algorithm can easily be adapted to solve the equation

$$u'(x) = 0 \tag{62}$$

instead of $u(x) = 0$, facilitating the computation of n local extreme values of u in order n operations.

Remark 9 In the algorithm described above, Taylor expansions are used to evaluate u close to its roots. Of course, Taylor expansions can also be used to evaluate u between roots, thus providing a scheme for the fast evaluation of u. Specifically, given a set of n points $\tilde{x}_1, \tilde{x}_2, \ldots, \tilde{x}_n$, where $\tilde{x}_{i-1} \leq \tilde{x}_i$, and m roots $x_1, x_2, \ldots, x_n$ of u, where $x_{i-1} \leq x_i$, the scheme finds the closest root to each point $\tilde{x}_i$ and evaluates u at $\tilde{x}_i$ via a Taylor expansion at that root. Including the cost for computing the m roots of u, the total cost of this scheme is of the order of $m + n$ operations.

3.2 Detailed description

In this section we present the algorithm in some detail. Algorithm 1 describes the core algorithm. As input, it requires the smallest root which is to be computed and the number N of roots to compute. It returns the roots $x_1 \ldots x_N$ and the derivatives $u'(x_1) \ldots u'(x_N)$ for a cost of order N operations. Algorithms 2 and 3 are auxiliary schemes for finding the initial root.

Algorithm 1
Comment[The input parameters are the polynomial coefficients of p, q, and r in equation (1), an initial root x_1 of u, the number of roots N which are greater than or equal to x_1, and the derivative at the initial root $u'(x_1)$. Algorithm 1 returns the N roots of u which are greater than or equal to x_1 in the vector *roots*. Furthermore, it returns the value of u' at the roots in the vector *ders*.]

Set $roots(1) = x_1$ and $ders(1) = u'(x_1)$.

do $i = 1,\ N - 1$

1. Use the Runge-Kutta method (20) to solve equation (12) for $\theta = \pi/2$ to $-\pi/2$ with initial conditions $\theta_0 = \pi/2$ and $x(\theta_0) = roots(\text{i})$. The resulting value for $x(-\pi/2)$ is the first approximation to $roots(i+1)$.

2. Use recurrence relation (18) to calculate the first 30 derivatives of u at $x = roots(i)$. Start the recurrence with $u(x) = 0$, $u'(x) = ders(i)$.

3. Increase the precision of $roots(i+1) = x(-\pi/2)$ via Newton's method (25), where u and u' are evaluated via the first 30 terms of the Taylor expansion around $roots(i)$. Set $ders(i+1) = u'(roots(i+1))$.

end do

If a local extremum of u is given an analogous procedure to steps 1-3 of Algorithm 1 can be used to calculate the next biggest root. This procedure is described in Algorithm 2.

Algorithm 2
Comment [The input parameters are the polynomial coefficients of p, q, and r in equation (1), an extreme value x_e of u, and $u(x_e)$. Algorithm 2 returns x_1, the smallest root of u which is bigger than x_e, and the derivative $u'(x_1)$.]

1. Use the Runge-Kutta method (20) to solve equation (12) for $\theta = 0$ to $-\pi/2$ with initial conditions $\theta_0 = 0$ and $x(\theta_0) = x_e$. The resulting value for $x(-\pi/2)$ is the first approximation to x_1.

2. Use recurrence relation (18) to calculate the first 30 derivatives of u at x_e. Start the recurrence with $u(x_e)$, $u'(x_e) = 0$.

3. Increase the precision of $x_1 = x(-\pi/2)$ via Newton's method (25), where u and u' are evaluated via the first 30 terms of the Taylor expansion around x_e.

If the first roots of u lie in a regime where the Taylor expansion has a poor convergence rate (see, for example, the results on Laguerre polynomials in Section 4.3), the following algorithm can be used.

Algorithm 3
Comment [Algorithm 3 requires a routine for evaluating u and u'. Its input parameters are the polynomial coefficients of p, q, and r in equation (1), and a starting

value x_s which is less than the smallest root. Algorithm 3 returns x_1, the smallest root of u which is bigger than x_s and $u'(x_1)$.]

1. Compute $\theta(x_s)$ via equation (10).
2. Use the Runge-Kutta method (20) to solve equation (12) for $\theta = \theta(x_s)$ to $-\pi/2$ with initial conditions $\theta_0 = \theta(x_s)$ and $x(\theta_0) = x_s$. The resulting value for $x(-\pi/2)$ is the first approximation to x_1.
3. Increase the precision of $x_1 = x(-\pi/2)$ via Newton iterations, where u and u' are evaluated by the given routine, which is specific to u. Return x_1 and $u'(x_1)$.

4 Implementation and numerical results

We have implemented the algorithm of this paper in FORTRAN 77 and tested it on several classical families of special functions. Below we discuss our implementation for Legendre, Hermite, and Laguerre polynomials, Bessel functions, and prolate functions. For each example we give timings and accuracy for different problem sizes. The computation time has been measured on an Intel Pentium 4, 2.4 GHz with 1 GB RAM. The given accuracy is based on computations in double precision.

The accuracy shown in the tables has been measured by computing the roots in double (15-digit) and in extended (30-digit) precision. Given N roots $x_1^{(d)} \ldots x_N^{(d)}$ of the function u in double precision and the exact values (calculated in extended precision) $x_1^{(q)}, \ldots, x_N^{(q)}$, the absolute accuracy r_a and the relative accuracy r_r are computed as

$$r_a = \max_{i \in \{1,\ldots,N\}} \left| x_i^{(d)} - x_i^{(q)} \right|, \tag{63}$$

$$r_r = \max_{i \in \{1,\ldots,N\}} \left| \frac{x_i^{(d)} - x_i^{(q)}}{x_i^{(q)}} \right|. \tag{64}$$

Remark 10 One of the principal applications of the roots of orthogonal polynomials is numerical integration, as given by formula (39). In the tables for the orthogonal polynomials (Tables 1, 2, and 3), we show the relative accuracy of the roots and the absolute accuracy of the weights, as computed by the algorithm of this paper. In addition, we provide the accuracy of the quadrature formula (39).

For an orthogonal polynomial of degree n we measure this accuracy by using (39) to integrate the polynomial of degree $\frac{3}{2}n$ of the same family, e.g., the accuracy of 1000 computed Laguerre nodes is tested by integrating the Laguerre polynomial of degree 1500. The analytical value of this integral is zero, so that the resulting numerical value is a direct measure for the error.

The orthogonal polynomial which is to be integrated is evaluated via the procedure described in Remark 9. To avoid additional numerical errors, these computations are performed in extended precision (30 digits).

4.1 Legendre polynomials

Equation (27) is the ODE defining the n^{th} degree Legendre polynomial P_n. Due to the symmetry of Legendre polynomials it is sufficient to compute only the positive roots. If n is even, P_n has an extreme value at $x = 0$, and Algorithm 2 is used to find the first positive root, followed by Algorithm 1 to find the $n/2 - 1$ subsequent roots. If n is odd, P_n has a root at $x = 0$, and Algorithm 1 is used to find the $(n-1)/2$ positive roots. The values for $P_n(0)$ or $P_n'(0)$ which are required as input for Algorithm 2 or 1 are calculated via the recurrence relations (32) and (33). Finally, the Legendre quadrature weights are calculated via formula (40).

Table 1 shows the total computation time, as well as the accuracy of the roots, the weights, and the quadrature formulae (see Remark 10). The total computation time includes the computation of the quadrature weights.

4.2 Hermite polynomials

To avoid numerical overflow, we have applied our algorithm to the Hermite functions $\widetilde{H}_n$ instead of the polynomials (see Remark 3). The differential equation for the Hermite functions is given in (48).

Since Hermite polynomials have the same symmetry as Legendre polynomials, the computation of the first root and the application of the main algorithm is completely analogous. The values $\widetilde{H}_n(0)$ and $\widetilde{H}_n'(0)$ can be calculated via the recurrence relations (50), (51). Formula (52) is used to calculate the Hermite weights from the returned roots and derivatives of the Hermite functions. Table 2 shows computation time and accuracy (see Remark 10).

4.3 Laguerre polynomials

As in the Hermite case, we have used the differential equation for the Laguerre functions given in equation (49) instead of the ODE for the Laguerre polynomials (see Remark 3). Equation (38) provides the starting value $x_s = \frac{2}{4n+2}$, and recurrence relations (53) and (54) are used for the evaluation of the function and its derivative. The quadrature weights are calculated via formula (55). Table 3 shows timings and accuracy (see Remark 10).

Remark 11 The differential equations (29), (49) have a singularity at $x = 0$, leading to bad convergence rates of the Taylor expansion of $\widetilde{L}_n$ close to $x = 0$. In order to avoid this issue, the first 20 roots are calculated via repeated application of Algorithm 3; the subsequent roots are then calculated via Algorithm 1.

4.4 Bessel functions

The differential equation for the n^{th} order Bessel function J_n of the first kind is given in equation (56). The first root is found via Algorithm 3 with the starting value $x_s = n + n^{\frac{1}{3}}$ as given by equation (57). The values for $J_n(x_s)$ and $J'_n(x_s)$ are calculated by means of a standard scheme, which can, for example, be found in 9.12, 9.1.27, [1].

We compute the first $20\,n$ roots of J_n. For larger roots, J_n will start to behave like a cosine, due to the asymptotic formula (58), and the distance between two consecutive roots approaches π. Computation time and accuracy for the Bessel roots are shown in Table 4.

4.5 Prolate spheroidal wave functions

The differential equation for prolate spheroidal wave functions is given in equation (61). We refer the reader to [13, 10] for a procedure to calculate the parameter χ_n^c. The computation of the first root and the application of the algorithm to prolate functions is completely analogous to Legendre polynomials, since they have same symmetry. Table 5 shows computation time and accuracy.

Acknowledgments

We would like to thank Dr. Mark W. Tygert for many useful discussions and suggestions.

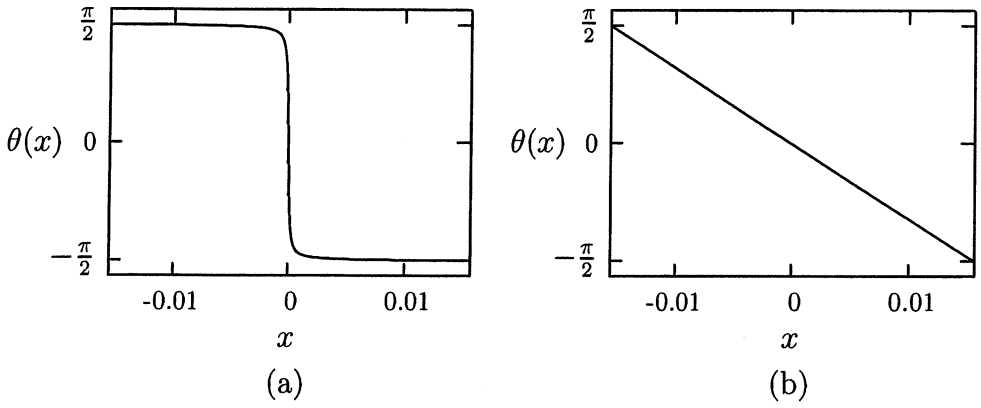

Figure 1: Figure (a) shows θ as defined in equations (2) and (3) for a Legendre polynomial of order 100 with $\gamma \equiv 1$. Figure (b) shows θ for the same polynomial with $\gamma \equiv \sqrt{rp}$.

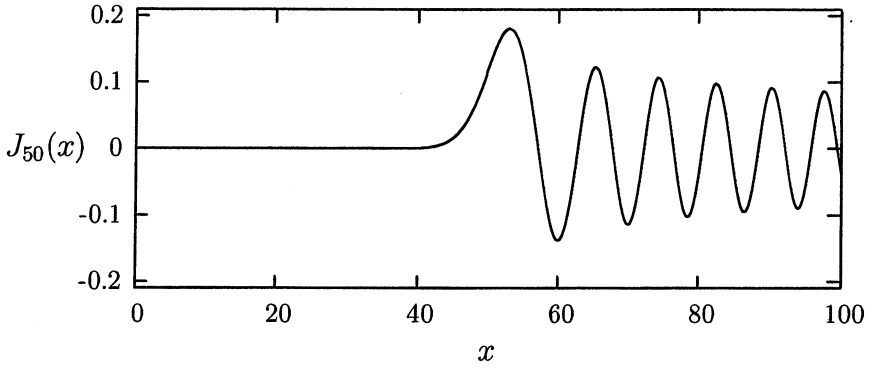

Figure 2: Plot of J_{50}

number of nodes	relative acc. of roots	absolute acc. of weights	accuracy of quadrature	computation time in sec.
1 000	4×10^{-15}	2×10^{-16}	6×10^{-16}	0.02
10 000	5×10^{-15}	1×10^{-16}	4×10^{-15}	0.15
100 000	7×10^{-15}	1×10^{-16}	2×10^{-15}	1.30
1 000 000	3×10^{-14}	5×10^{-17}	2×10^{-15}	12.31

Table 1: Timings and accuracy for Legendre roots and weights (see Remark 10).

number of nodes	relative acc. of roots	absolute acc. of weights	accuracy of quadrature	computation time in sec.
1 000	5×10^{-15}	2×10^{-16}	2×10^{-16}	0.02
10 000	1×10^{-14}	1×10^{-16}	2×10^{-15}	0.16
100 000	5×10^{-14}	3×10^{-16}	6×10^{-16}	1.54
1 000 000	9×10^{-13}	9×10^{-17}	2×10^{-16}	15.36

Table 2: Timings and accuracy for Hermite roots and weights (see Remark 10).

number of nodes	relative acc. of roots	absolute acc. of weights	accuracy of quadrature	computation time in sec.
1 000	4×10^{-12}	2×10^{-13}	1×10^{-13}	0.04
10 000	4×10^{-10}	9×10^{-13}	2×10^{-13}	0.37
100 000	8×10^{-8}	1×10^{-11}	8×10^{-12}	3.27
1 000 000	2×10^{-6}	1×10^{-10}	5×10^{-11}	31.18

Table 3: Timings and accuracy for Laguerre roots and weights (see Remark 10).

number of computed roots	order n of Bessel function	relative acc. of roots	computation time in sec.
2 000	100	5×10^{-15}	0.06
20 000	1 000	6×10^{-15}	0.58
200 000	10 000	3×10^{-14}	5.83
2 000 000	100 000	9×10^{-14}	58.20

Table 4: Timings and accuracy for the roots of Bessel functions.

number of computed roots	band-limit c	relative acc. of roots	computation time in sec.
2 000	1 000	4×10^{-15}	0.03
20 000	10 000	6×10^{-15}	0.28
200 000	100 000	2×10^{-14}	2.33
2 000 000	1 000 000	4×10^{-14}	22.27

Table 5: Timings and accuracy for the roots of prolate spheroidal wave functions.

References

[1] M. ABRAMOVITZ AND I. A. STEGUN, eds., *Handbook of Mathematical Functions*, Dover Publications, Mineola, New York, 1972.

[2] J. S. BALL, *Automatic computation of the zeros of Bessel functions and other special functions*, SIAM Journal on Scientific Computation, 21 (1999), pp. 1458–1464.

[3] G. BIRKHOFF AND G.-C. ROTA, *Ordinary Differential Equations*, Ginn and Company, 1962, ch. 10.5.

[4] G. DAHLQUIST AND Å. BJÖRK, *Numerical Methods*, Dover Publications, Mineola, New York, Dover ed., 2003.

[5] W. GAUTSCHI, *Orthogonal Polynomials: Computation and Approximation*, Oxford University Press, 2004.

[6] G. H. GOLUB AND J. H. WELSCH, *Calculation of Gauss quadrature rules*, Mathematics of Computation, 23 (1969), pp. 221–230.

[7] M. GU AND S. EISENSTAT, *A divide-and-conquer algorithm for the symmetric tridiagonal eigenproblem*, SIAM Journal on Matrix Analysis and Applications, 16 (1995), pp. 172–191.

[8] Y. IKEBE, Y. KIKUCHI, AND I. FUJISHIRO, *Computing zeros and orders of bessel functions*, J. Comput. Appl. Math., 38 (1991), pp. 169–184.

[9] W. H. PRESS, S. A. TEULKOLSKY, W. T. VETTERLING, AND B. P. FLANNERY, *Numerical Recipes in Fortran 77*, Press Syndicate of the University of Cambridge, 2001, ch. 4.5.

[10] V. ROKHLIN AND H. XIAO, *Approximation formulae for certain prolate spheroidal wave functions valid for large values of both order and band-limit*, Tech. Rep. YALEU/DCS/RR-1323, Yale University, May 2005.

[11] G. SZEGÖ, *Orthogonal Polynomials*, American Mathematical Society, Providence, Rhode Island, 4th ed., 1978.

[12] M. TYGERT, *Recurrence relations and fast algorithms*, Tech. Rep. YALEU/DCS/TR-1343, Yale University, 2006.

[13] H. XIAO, V. ROKHLIN, AND N. YARVIN, *Prolate spheroidal wavefunctions, quadrature and interpolation*, Inverse Problems, 17 (2001), pp. 805–838.