Numerical Algorithms (2024) 96:1109–1141
https://doi.org/10.1007/s11075-023-01738-z

ORIGINAL PAPER

An Hermite–Obreshkov method for 2nd-order linear initial-value problems for ODE

with special attention paid to the Mathieu equation.

Robert M. Corless[1,2]

Received: 22 August 2023 / Accepted: 21 December 2023 / Published online: 9 January 2024

Abstract
The numerical solution of initial-value problems (IVP) for ordinary differential equations (ODE) is at this time a mature subject, with many high-quality codes freely available. Second-order linear equations without singularities are an especially simple class of problems to solve, even more so if only a single scalar equation such as the Mathieu equation $y'' + (a - 2q\cos 2x)y = 0$ is being considered. Nonetheless, the topic is not yet exhausted, and this paper considers the case of writing an efficient arbitrary-precision code for the solution of such equations. For this purpose, an implicit Hermite–Obreshkov method attains nearly spectral accuracy at a cost only polynomial in the number of bits of accuracy requested. This is interesting for the Mathieu equation in particular because the solutions can be highly oscillatory of variable frequency and be highly ill-conditioned. This paper reports on the details of the prototype Maple implementation of the method and summarizes the approximation theoretic results justifying the choice of a balanced Hermite–Obreshkov method including its backward stability and decent Lebesgue constants. This method may be of especial interest for the solution of so-called D-finite equations, for which Taylor series coefficients up to degree m are available at cost only $O(m)$, instead of the more usual $O(m^2)$. This paper celebrates the happy occasion of the 90th birthday of John C. Butcher.

Keywords Hermite–Obreshkov · Arbitrary-precision · Initial-value problems for ordinary differential equations

Mathematics Subject Classification (2010) 65L04 · 33F05 · 65D15

✉ Robert M. Corless
rcorless@uwo.ca

1 The Ontario Research Center for Computer Algebra, Department of Computer Science, and the Rotman Institute of Philosophy, Western University, London, ON, Canada

2 Cheriton School of Computer Science, University of Waterloo, Waterloo, ON, Canada

1 Introduction

I give here a detailed description of an implicit method using piecewise high-order Hermite interpolants for high precision, in fact, *arbitrary* precision, numerical solution of certain smooth initial-value problems (IVPs) for ordinary differential equations (ODEs). These methods are known in the literature as Hermite–Obreshkov methods, in honor of the nineteenth-century French mathematician Charles Hermite who first described osculatory interpolational polynomials, and for the twentieth-century Bulgarian mathematician Nikola Obreshkov who developed a quadrature formula using them. A significant potential advantage to this implicit approach compared to explicit Taylor series methods using m terms at each step is that, for only modestly more computation on linear problems, the implicit method is $O(h^{2m})$ accurate instead of being only $O(h^m)$ accurate. This can result in significantly better efficiency even for some nonstiff problems.

The idea is that if one has gone to the trouble of generating Taylor coefficients at both ends of the interval, one might as well use both sets of Taylor coefficients to approximate the solution over the step.

Similar methods using Taylor series, including implicit methods, have already been implemented and described for general-purpose use for the interval solution of IVP [35] and (in standard precision point arithmetic) of differential-algebraic equations (DAEs) in [36], and moreover those authors have made their DAETS code freely available. Then, there is the recent paper [47] which even solves nonlinear problems using a Hermite–Obreshkov method. See also [41] which uses projection, optimization, and implicit methods (also for DAE). Since those codes and their methods are much more general than the code that I will describe, the reader may wonder why they should read further here.

More, in a sequence of papers, namely [8, 14, 15, 18] I and co-authors have already sketched a great deal about this niche method and its applications to the computation of Mathieu functions and thereby to the blood flow application that originally motivated us. A reader "skilled in the art" could instead read all those papers instead of this one and glean or deduce many of the things that I will discuss here.

There are several novel points about this algorithm, however, which this present paper brings out. For one, unlike the methods in the cited papers, the method I describe uses *collocation* for the direct solution of (linear) higher-order equations without conversion to a first-order system. Although direct collocation for higher-order equations is not itself new, it is novel in this context. This direct use of collocation for higher-order equations enables the use of the residual (also called the *defect*) for error control, which makes error analysis in the context of the original problem simpler, especially for the non-specialist. In Sect. 3.5, I will explain this further.

I use a new and numerically stable version of the explicit form of the two-point Hermite interpolational polynomial described in 1873 by Hermite, unlike [47] which uses a divided-difference formulation. I give here full details of the stability analysis, which was only sketched in one of the previous papers, namely [15]. I also give a wholly new and numerically stable version of Obreshkov's quadrature formula.

Finally, I describe the use of the method on so-called D-finite differential equations, which are explained below and for which the method ought to be particularly attractive.

1.1 Arbitrary-precision computation

Something else that might be novel to this audience is the genuine need for arbitrary-precision computation. This is becoming increasingly important in certain modern applications [4]. The main point of [4] is that for some problems, it is simply more efficient to use brute force and high precision to overcome numerical difficulties such as that the problem is ill-conditioned or that the available algorithm is numerically unstable. Some scientists are unable or unwilling to develop new and more stable algorithms, even if such exist for the problem at hand. In some cases, it is simply more efficient in terms of human time to use arbitrary-precision computation, even if it is vastly more expensive in terms of computing time than (say) double precision. Admittedly this is not that common yet; all that those authors are saying is that *sometimes* this is useful, and may be more useful in the future.

For the case of the modified Mathieu equation $y'' - (a - 2q\cosh 2x)y = 0$, some solutions grow *doubly exponentially* with x, and oscillate with exponentially increasing frequency as well. The most numerically stable algorithm available will still demand high precision even if one only wants to know (say) the correct sign of the real part of the function for large x.

Some numerical notions change in this context, while others remain valid. For instance, one might happily use a mildly unstable polynomial as an approximant for a function, and simply add a few more guard digits in the computation. This typically increases the cost of computation only by a negligible marginal amount. But, if the approximant is *exponentially* unstable, then that is a different matter and even in an arbitrary-precision environment, one must take care. I will show in this paper that we may use Hermite interpolational polynomials of degrees up to a thousand or so—which is usually far more than enough—with very little numerical instability even just in ordinary double precision, in some situations, by using the right method. Using a naive method instead would cause problems already by degree 30 or so. In other situations, the numerical instabilities are unavoidable. Some recent results in the approximation theory of piecewise Hermite interpolation are helpful here [14, 18]. One thing omitted in those references is a discussion of the rate of convergence of these approximations, and we sketch this in Sect. 2.6.

In the context of arbitrary-precision computation for smooth functions, very high-order methods are the most useful. This is because, technically speaking, fixed-order methods are of cost exponential in the number of bits of accuracy requested. Indeed the lowest-order methods are the most expensive. In contrast, arbitrary order methods are of cost polynomial in the number of bits of accuracy requested [32].

1.2 This method uses derivatives (Taylor coefficients)

One dominant viewpoint of numerical methods for IVP is that using function values (the $\mathbf{f}$ in the ODE $\mathbf{y}' = \mathbf{f}(\mathbf{y})$) is cheaper than using derivatives or Taylor coefficients. This viewpoint drives multistep methods, Runge–Kutta methods, and general linear methods. The reasoning goes that the computation of derivatives, especially for high-dimension $\mathbf{f}$, gets expensive very quickly.

This is not the only viewpoint, and there has been significant research on the value of using even just one more derivative, e.g., [11, 21]. In addition to the previously cited papers on Hermite–Obreshkov methods there is also a thriving literature on Taylor series methods. See, for instance, [19] or more recently [1].

These methods all work well for *smooth* problems. Discontinuities, even discontinuities in the derivatives, require special handling. The methods are also typically harder to code because the automatic generation of Taylor coefficients is somewhat involved. In compensation, the generation of those coefficients gives free interpolants of high quality, which we will use to advantage here.

Finally, we call attention here to what are called D-finite functions or holonomic functions [31, 33, 34]. These are defined to be the solutions of linear homogeneous differential equations with polynomial coefficients. They are simple, but they occur surprisingly frequently. Such an ODE has a linear recurrence relation for its Taylor coefficients of a *fixed* length, unlike the normal case where the recurrence for the degree m coefficient involves all m of the previous coefficients. Of course, the length of the recurrence depends on the degrees of the polynomial coefficients. If the differential equation is D-finite, then the Taylor coefficients at an arbitrary point can be generated with $O(m)$ cost instead of $O(m^2)$ cost. For such problems, Taylor series methods and Hermite–Obreshkov methods become even more attractive.

I point out for clarity that the usual metric for the cost of a method, namely to count the number of function evaluations it requires, does not obtain for these methods, and does not obtain for the methods discussed here. Instead, one tries to measure the cost of the construction of the Taylor coefficients at each node, which is typically the dominant cost of a step and replaces the idea of the cost of the function evaluations. For a general system, this is (as stated previously) $O(m^2)$, because the cost of computing the Cauchy products involved is $O(m^2)$ (unless one uses the FFT, which most codes do not do, but see [6] for a possible approach). This construction has to be done for each component (typically all at once), so for a problem of dimension d the cost will be $O(dm^2)$. Larger m will generally allow larger stepsizes because the problems are smooth, and as usual, we will want to equidistribute the error by choosing the stepsizes to make the error, however measured, approximately equal to (no greater than) the user's tolerance.

The question of variable *order* will be discussed briefly in the concluding remarks. I do not use variable order here, because it was not needed for the applications I had in mind.

1.3 Outline of the paper

In Sect. 2, I give the basic formula, known already to Hermite, for a smooth interpolational polynomial given Taylor coefficients at either end of an interval—called here a "blend"—and discuss its rapid and numerically stable evaluation, differentiation, and integration using exact quadrature. In Sect. 2.6, I summarize the convergence theory for these approximations. In Sect. 2.5, I describe the construction of a piecewise spline-like interpolant using blends. Blends need Taylor series coefficients, which limits their use to smooth functions, and I discuss automatic generation from differential equations

including so-called D-finite equations in Sect. 3.1. The solution of stiff or oscillatory differential equations benefits from some degree of implicitness, and I discuss the use of collocation for this, giving a variable stepsize method that approximately equidistributes the residual or "defect," in Sect. 3.4. The numerical stability of the resulting implicit method—in short, it's not A-stable but it's not bad either—is discussed in detail in Sect. 4. I give the results from some numerical experiments in Sect. 5 and conclude in Sect. 6.

2 Blends, aka two-point Hermite interpolational polynomials

Much of this section is based on [18]; an open version of that paper is available at https://arxiv.org/abs/2007.05041, if the reader wishes more detail.

Approximation of a function $f(z)$ by its Taylor polynomials at a single point $z = a$, namely $P_N(z) = \sum_{j=0}^{N} p_j (z-a)^j$ where each $p_j = f^{(j)}(a)/j!$, is a staple part of every undergraduate course in approximation theory. The Lagrange form of the error

$$f(z) - P_N(z) = \frac{f^{(N+1)}(\theta)}{(N+1)!}(z-a)^{N+1} \tag{1}$$

where θ is some number between z and a is part of many first-year calculus courses. On an interval of width h this gives an error of $O(h^{N+1})$ with an error of $O(h^N)$ in the derivative: $f'(z) - P_N{}'(z) = O(h^N)$.

It is less well-known that one can use the Taylor coefficients at a second point $z = b$, namely $q_j = f^{(j)}(b)/j!$, to give a much-improved approximation on the interval $a \le z \le b$. Without loss of generality, we assume $a < b$, and later we will consider the case of a line segment joining two complex points a and b. For now, we assume the variables are real. For ease of notation, we make a change of variables to $s = (z-a)/(b-a)$ so that $z = a + s(b-a)$. Then, the following formula was known already to Hermite [29]:

$$\begin{aligned} H_{m,n}(s) = &\sum_{j=0}^{m} \left[\sum_{k=0}^{m-j} \binom{n+k}{k} s^{k+j} (1-s)^{n+1} \right] p_j \\ &+ \sum_{j=0}^{n} \left[\sum_{k=0}^{n-j} \binom{m+k}{k} s^{m+1} (1-s)^{k+j} \right] (-1)^j q_j \end{aligned} \tag{2}$$

We say that a polynomial has *grade* m if its degree is at most m. This is convenient in cases, as here, where the leading coefficient is hidden and might be zero. The formula above is of grade $m + n + 1$ and has Taylor coefficients (in the s variable—one has to be careful to use the chain rule to change coefficients to and from the original z variable) $p_j = f^{(j)}(0)/j!$ and $q_j = f^{(j)}(1)/j!$.

Definition of a blend. $H_{m,n}(s)$ is a *two-point Hermite interpolational polynomial* in that it satisfies the given Hermite interpolational data at either end. This term is

too complicated to say easily, and so I use the word "blend" instead. This seems apt because the polynomial "blends" together the Taylor coefficients at either end.

The Lagrange form of the error in a blend is analogous to the Lagrange form of the error in Taylor approximation:

$$f(s) - H_{m,n}(s) = \frac{f^{(m+n+2)}(\theta)}{(m+n+2)!} s^{m+1}(s-1)^{n+1} \tag{3}$$

for some $\theta = \theta(s)$ between 0 and 1. On the interval $0 \le s \le 1$ the polynomial $s^{m+1}(1-s)^{m+1}$ is 2^{2m+1} times smaller than the polynomial s^{2m+1}, which is the corresponding term in the Lagrange form of the error for Taylor approximation. Two-point Hermite interpolation can therefore be substantially more accurate on the interval than Taylor approximation of equivalent grade. On an interval of width h this gives an $O(h^{m+n+2})$ error, with one less power of h for every derivative taken.

A complex-valued version of this error formula (due to Hermite) is available:

$$f(s) - H_{m,n}(s) = \frac{1}{2\pi i} \oint \frac{s^{m+1}(1-s)^{n+1} f(\zeta)}{\zeta^{m+1}(1-\zeta)^{n+1}(\zeta - s)} \, d\zeta \ , \tag{4}$$

where the counterclockwise integration takes place on a closed contour that encloses s, 0, and 1, and no singularities of f. We give more details about this in Sect. 2.6.

There is also the principle of economy: if you are going to generate Taylor series of grade m at every point in a numerical solution of a differential equation, you might as well use those at either end of the interval to interpolate with.

The numbers $\binom{n+k}{k}$, for $0 \le k \le m$, and identically $\binom{m+k}{k}$, $0 \le k \le n$, which appear in formula (2), grow large rather quickly. This may (should!) alarm numerical analysts because their exponential growth may cause instability in the numerical evaluation of the formula.

2.1 Evaluation by Horner's method is fast and stable

However, by rewriting the formula in Horner form, noting that the basis functions can be constructed in a single loop, and simultaneously using the recurrence relation

$$\binom{n+k}{k} = \frac{n+k}{k} \binom{n+k-1}{k-1} \tag{5}$$

we can defuse the potential instability. Using this method, Algorithm 1 implements this idea. Both of the sums in the Hermite formula can be evaluated by interchanging s and $1-s$ and m and n and inserting the correct signs, as necessary. The total cost for the evaluation, measured in flops, is $O(m+n)$. In an arbitrary-precision floating-point environment, each of those "flops" might be more expensive than a double-precision flop, but (for instance) Maple is IEEE-854 compliant and at least the results are reliable, and of a cost more or less fixed once the precision has been chosen.

Algorithm 1 Half Sum, without derivatives.

```
1: procedure HALFSUM(σ, m, n, w)          ▷ m,n nonnegative integers, w Array(0,m)
2:     a_0 ← 1                             ▷ Accumulator loop
3:     for k from 1 to m do                ▷ n gets used in the sum
4:         a_k ← (n + k) · σ · a_{k−1}/k   ▷ 0 ≤ σ ≤ 1
5:     end for
6:     s_0 ← a_0                           ▷ s_k are partial sums of nonnegative numbers
7:     for k from 1 to m do
8:         s_k ← s_{k−1} + a_k
9:     end for
10:    U ← 0
11:    for j from m by −1 to 0 do          ▷ Coefficients w_j used here
12:        U ← s_{m−j} · w_j + σ · U
13:    end for
14:    C ← 1
15:    for j from 1 to n + 1 do            ▷ The complementary factor is (1 − σ)^{n+1}
16:        C ← (1 − σ) · C
17:    end for
18:    S ← C · U
19:    return S                            ▷ The half sum is S
20: end procedure
```

We then have the following numerical stability theorem [14]:

Theorem 1 *For $0 \le s \le 1$ and with real Taylor coefficients p_j and q_j, Algorithm 1 produces the exact sum for slightly different Taylor coefficients, namely $\mathrm{fl}(p_j) = p_j(1+\theta_{3m-j+n+4})$ and $\mathrm{fl}(q_j) = q_j(1+\theta_{3n-j+m+4})$. This implies that this algorithm applied to a blend produces the exact value of a blend with coefficients differing at most by a factor $1+\gamma_{\max(3m+n,3n+m)+4}$ from their inputs.*

We use the notation of Lemma 3.1 of [30], where $\gamma_j = ju/(1-ju)$ with u being the unit roundoff. Theorem 1 guarantees an extremely strong backward stability result, namely that the algorithm gives the exact value of a blend where the coefficients are changed *componentwise* only by a tiny relative amount: essentially only a linear number of rounding errors, and zero coefficients are undisturbed. This is quite comparable to similar componentwise stability results for Chebyshev polynomial expansion [44]. One could hardly ask better, and it is a relief that the potentially large binomial coefficients are not a problem after all.

2.2 Approximation by balanced blends is accurate

One then has to worry about whether these interpolational polynomials are at all sensitive to changes in their coefficients. The answer to that depends on just where the polynomial is being evaluated. To explain this most simply, we use the notion of the Lebesgue function and Lebesgue constant. If

$$f(s) = \sum_{j=0}^{N} c_j \phi_j(s) \tag{6}$$

is a polynomial expanded in some basis $\phi_j(s)$, and has coefficients perturbed to $c_j(1+\varepsilon_j)$ with all $|\varepsilon_j| \le \varepsilon$, then

$$\begin{aligned} |\Delta f(s)| &= \left| \sum_{j=0}^{N} c_j \varepsilon_j \phi_j(s) \right| \\ &\le \sum_{j=0}^{N} |c_j| \left|\phi_j(s)\right| \|\varepsilon_j\|_\infty \\ &\le L(s) \|c_j\|_\infty \varepsilon \end{aligned} \tag{7}$$

where $L(s) = \sum_{j=0}^{N} |\phi_j(s)|$ is the *Lebesgue function* and the Lebesgue *constant* is (for us, with our doubly-indexed expansion)

$$L_{m,n} := \max_{0 \le s \le 1} L_{m,n}(s) \,. \tag{8}$$

For blends, the Lebesgue function turns out to itself be a blend, where all the coefficients $p_j = 1$ and all the coefficients $q_j = (-1)^j$, because each basis function is a sum of nonnegative terms and is therefore itself nonnegative, on $0 \le s \le 1$. This allows one to prove that for balanced blends $L_{m,m} < 2$ if $0 \le s \le 1$.

Therefore, the forward error is at most twice the backward error, which we have shown to be small. See [14] for details.

One important point is that grossly unbalanced blends can have exponentially-growing Lebesgue constants, if m and n are very different, say $m < n/4$ or vice-versa. Another is that the error grows very rapidly (like $|s|^{m+n+1}$) if s is at all distant from the interval [0, 1].

2.3 Semi-automatic differentiation of a blend

Derivatives of the interpolant are always useful. To this purpose it is straightforward to differentiate Algorithm 1; that is, to code it in such a fashion that it supplies not only $H_{m,n}(s)$ but also as many derivatives as are needed. The cost per derivative is essentially the same as the cost of evaluating the blend, $O(m+n)$. The same backward error theorem applies, and the resulting computed derivatives are the exact derivatives of a blend with relatively tiny numerical perturbations.

2.4 Exact quadrature of a blend

The following beautiful exact quadrature of a blend is very useful. Obreshkov has this formula written in terms of ratios of binomial coefficients in [37]. I think that the formula must have been known to Hermite, and seems to be in the writings of Darboux: see [20]. I proved it for myself using the contour integral method that John Butcher uses (see, e.g., [10]). The following is taken from [14, 18].

Computing the *definite* integral of a blend over the entire interval will allow us to construct a new blend whose value at any point is the *indefinite* integral of the original blend up to that point, namely

$$I(x) = \int_{s=0}^{x} H_{m,n}(s)\, ds\ . \tag{9}$$

The definite integral that we need is

$$I(1) = \int_{s=0}^{1} H_{m,n}(s)\, ds = \frac{(m+1)!}{(m+n+2)!} \sum_{j=0}^{m} \frac{(n+m-j+1)!}{(j+1)\,(m-j)!}\, p_j + \frac{(n+1)!}{(m+n+2)!} \sum_{j=0}^{n} \frac{(n+m-j+1)!}{(j+1)\,(n-j)!}\, (-1)^j\, q_j\ . \tag{10}$$

This is an exact complete integral across the subinterval if the coefficients are known exactly, but the main use of this routine is when the coefficients are floating-point numbers, in which case this becomes a kind of numerical quadrature.

The definite integral (10) allows one to (trivially) compute the Taylor coefficients for $I(s) = \int_0^s H(\sigma)\, d\sigma$ at $s = 1$: the zeroth order coefficient is now known (it's just the definite integral $\int_0^1 H(\sigma)\, d\sigma$) and all the derivatives are simply related to the (known) derivatives of $H(s)$ at $s = 1$:

$$\begin{aligned} I'(s) = H(s) &= \sum_{j=0}^{m} q_j (s-1)^j + O((s-1)^{m+1}) \\ I(s) &= I(1) + \sum_{j=0}^{m} \frac{q_j}{j+1} (s-1)^{j+1} + O((s-1)^{m+2})\ . \end{aligned} \tag{11}$$

Likewise, $I(0) = 0$ is known, and all its derivatives at $s = 0$ are related in the same way.

In [14], I showed that the error in approximating the integral of $f(s)$ by the integral of $H_{m,n}(s)$ is small if the error in approximating $f(s)$ by $H_{m,n}(s)$ is small. In particular, the use of balanced blends with m nearly equal to n is strongly recommended.

In detail, we have the following theorem.

Theorem 2 *If the coefficients of the blend are in error by at most Δp_j for $0 \le j \le m$ and Δq_j for $0 \le j \le n$, then the error in the integral of the blend is bounded by $\int_0^1 L_{m,n}(s)\, ds \max |\Delta p_j|, |\Delta q_j|$ where $L_{m,n}(s)$ is the Lebesgue function for the blend.*

Further, this bounding integral can be explicitly computed. We find

$$\int_0^1 L_{m,n}(s)\, ds = 2\Psi(n+m+3) - \Psi(m+3) - \Psi(n+3) + \frac{n+m+4}{(n+2)(m+2)}\ . \tag{12}$$

Here, $\Psi(n+1) = -\gamma + \sum_{k=1}^{n} 1/k$ is the logarithmic derivative of the factorial function.

If either n or m goes to infinity while the other remains fixed, this integral grows like $\ln n$ or $\ln m$. If both $n = m$ go to infinity together the integral is asymptotic to $2\ln 2 - 1/(2m) + O(1/m^2)$.

This shows that for balanced blends, the computation of the integral by using this formula is numerically stable, and that is certainly what is observed in practice.

2.4.1 Computational version

Put $c_0 = (m+1)/(m+n+2)$ and $d_0 = (n+1)/(m+n+2)$ and define

$$c_j = \frac{j(m-j+1)}{(j+1)(m+n+2-j)} c_{j-1} \qquad \text{for} \qquad 1 \le j \le m \tag{13}$$

$$d_j = -\frac{j(n-j+1)}{(j+1)(m+n+2-j)} d_{j-1} \qquad \text{for} \qquad 1 \le j \le n\,. \tag{14}$$

Notice the sign alternation in the d_js. Then,

$$I(1) = \sum_{j=0}^{m} c_j p_j + \sum_{j=0}^{n} d_j q_j\,. \tag{15}$$

If $m = n$ then $|d_j| = c_j$ and the formula becomes

$$\begin{aligned} I(1) &= \sum_{j=0}^{m} c_j \left(p_j + (-1)^j q_j\right) \\ &= \frac{1}{2}(p_0 + q_0) + \frac{m}{4(2m+1)}(p_1 - q_1) + \cdots + c_m\left(p_m + (-1)^m q_m\right)\,. \end{aligned} \tag{16}$$

The final coefficient $c_m = 1/((m+1)\binom{2m+2}{m+1})$ is asymptotic to $2^{-2m-2}\sqrt{\pi/m}$. A short analysis shows that his method is *beautifully* stable numerically: the componentwise backward error using IEEE 854 standard floats is bounded by γ_{m+n+2}.

The method is not very *accurate* if m and n are greatly different, but if $m \approx n$ it's wonderfully accurate when the underlying function has no nearby singularities. Notice also that if $m = n = 1$ then this is the well-known corrected trapezoidal rule.

2.5 Blendstrings

Given a sequence of distinct points (or "knots") $z_0, z_1, \ldots, z_N$ in the complex plane, at which Taylor coefficients up to grade m_k are known, it is obviously possible to construct blends on each line segment $z_k + s(z_{k+1} - z_k)$. Since the Taylor coefficients at each interior knot are specified, this piecewise polynomial interpolant is very smooth; if all m_k are the same, say m, then the interpolant is m-times continuously differentiable.

This smoothness has some desirable properties for approximation. For the Mathieu equation, we only needed to take the knots in a straight line; either on one of the real intervals $[0, \pi]$ or $[0, 2\pi]$, or on a purely imaginary line $[0, ix_f]$ for some real number x_f. But one can imagine other applications needing more complicated paths.

Given two compatible "blendstrings" as I call them (they are really just a high-order kind of piecewise polynomial Hermite interpolant, but that's a mouthful) one can add, subtract, multiply, or even divide them (so long as the dividend isn't zero at a knot; even if the denominator is zero only between knots division can be problematic, though). One can differentiate a blendstring, although it's probably best not to construct a new blendstring for the derivative. One can integrate a blendstring, using the quadrature formula on each piece. In this case, it is quite useful to construct a new blendstring for the indefinite integral. Producing a numerical solution to an ODE that one can treat as a first-class function in this way is very convenient.

One important feature is that the order of each blend will be compatible, as we shall see, with the order of the numerical method for solving the differential equation. Looking ahead, we will use the differential equation and the Taylor series coefficients we generate in a way that allows us to choose the knots so that the residual will be bounded by the user's tolerance; this essentially equidistributes the error along the path of integration.

2.6 Convergence of blendstrings

As described in [46, Chap. 11], the convergence of an (m, m) blend to the underlying function $f(s)$ is—as with all polynomial approximations using data on the interval—exponential in m if the distance to the nearest singularity is large enough. Specifically, if the contour in (4) can be taken so that every point ζ on it has

$$\left|\frac{s(1-s)}{\zeta(1-\zeta)}\right| \le \frac{1}{\rho} \tag{17}$$

for some $\rho > 1$ and any $s \in [0, 1]$, then the convergence is $O(\rho^{-m})$ for a balanced (m, m) blend. This is because

$$\left|f(s) - H_{m,m}(s)\right| \le \frac{1}{2\pi} \oint \left|\frac{s^{m+1}(1-s)^{m+1}}{\zeta^{m+1}(1-\zeta)^{m+1}}\right| \left|\frac{f(\zeta)}{(\zeta - s)}\right| |d\zeta|\,, \tag{18}$$

$$\le \frac{M}{\rho^{m+1}} \tag{19}$$

for some constant M independent of m.

We can guarantee that (17) holds by insisting that $|\zeta(1-\zeta)| \ge \rho/4$, because $1/4 \ge s(1-s)$ for any $s \in [0, 1]$. Regions where this is true are shown in Fig. 1.

In particular, a pole close to either evaluation point ($s = 0$ or $s = 1$) gives more trouble than a pole close to $s = 1/2$ does. We find by experiment (not shown here) that the largest error appears in the middle, however, not near the ends.

If $f(z)$ is entire, then the convergence is even faster.

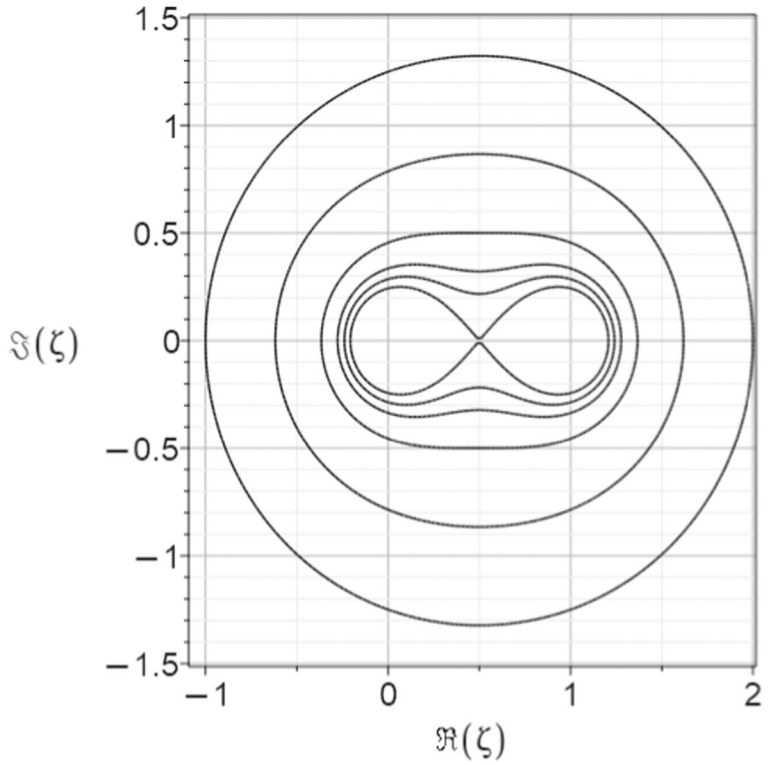

Fig. 1 Contours of $4|\zeta(1-\zeta)|$. The contours are, from the inside out, 1, $2^{1/4}$, $2^{1/2}$, 2, 4, and 8. The contour that intersects itself is at $4|\zeta(1-\zeta)| = 1$, and if $f(\zeta)$ has a pole inside that figure 8 then the balanced blends $H_{m,m}(s)$ will not converge to $f(s)$ on $0 \le s \le 1$ as $m \to \infty$. If all poles are outside a contour with value $\rho > 1$, then $H_{m,m}(s) - f(s)$ goes to zero at least as fast as $O(\rho^{-m})$ as $m \to \infty$

If, however, there is a singularity of $f(s)$ near to either end of the interval, then the sequence of blends might not converge at all.

The cure for this of course is to use blend*strings*. If we subdivide the interval up with a small enough mesh, then the *relative* distance to the singularity becomes large enough so that exponential convergence is recovered.

For blendstrings with subintervals of uniform width h, the error can be considered to be $O(h^{2m+2})$ as $h \to 0$. If the mesh is not uniform but the error is equidistributed, the error can be considered to be $O(\overline{h}^{2m+2})$ where $\overline{h}$ is the *arithmetic mean* stepsize [13]. This error model may not be terribly useful, because we normally want to have quite large mesh widths, and perhaps we will even want a large mean mesh width.

As discussed in [46, Chapter 11], blends are not as accurate as Chebyshev polynomial interpolation by a factor of 2^{m+n}; still, they are better than plain Taylor approximation, also by a factor of 2^{m+n}. In some sense, then, they are mid-way between the two, in terms of accuracy.

2.6.1 *Indefinite* integration: blendstring quadrature

Following the discussion of (11), we can immediately find a blend for $I(s)$ on $0 \leq s \leq 1$.

By propagating information from the previous subinterval (essentially adding the necessary constant of integration), this gives us a blendstring for the integral of the function being approximated by the blendstring.

Since integrals of expressions containing the solution to the IVP are frequently needed in applications, this feature is quite convenient.

3 A collocation method for solving IVP

3.1 Automatic generation of Taylor coefficients

"It's just code generation."

—Y.F. Chang, some time around 1990

Automatic differentiation is by now a fairly mature field. Early work includes [39] and by the 1990s effective and general-purpose codes were available [26]. For the scalar problems discussed here, a much simpler tool suffices, namely the routine `diffeqtorec` (for "differential equation to recurrence relation") from the `gfun` package in Maple [40].

For example, consider the Mathieu differential equation:

$$\frac{d^2}{dx^2} y(x) + (a - 2q\cos(2x))\, y(x) = 0\,. \tag{20}$$

If we change variables to v where $x = \arccos(c+v)/2$, the equation becomes (apparently) much simpler. This transformation, or, rather, ones like it, was known already to Mathieu.

$$4\left(1-(c+v)^2\right)\left(\frac{d^2}{dv^2}y(v)\right) - 4\,(c+v)\left(\frac{d}{dv}y(v)\right) + (a - 2q(c+v))\, y(v) = 0\,. \tag{21}$$

This equation is in D-finite form because the variable v only appears polynomially in the coefficients. Calling `diffeqtorec` on this equation results in the fixed-order recurrence relation

$$u_{k+3} = -\frac{q u_k}{2\,(k+3)\,(k+2)\left(c^2-1\right)} + \frac{\left(2qc - 4k^2 + a - 8k - 4\right) u_{k+1}}{4\,(k+3)\,(k+2)\left(c^2-1\right)} + \frac{(2k+3)\,c u_{k+2}}{\left(c^2-1\right)(k+3)}\,. \tag{22}$$

The solution to this recurrence relation gives the Taylor coefficients u_k for $y(v)$ around $v = 0$:

$$y(v) = \sum_{k\geq 0} u_k v^k\,. \tag{23}$$

Since I had transformed the differential equation to a new variable $c + v$ about a symbolic point c, this in fact gives the general recurrence relation for the Taylor coefficients of $y(v)$ about an arbitrary point, so long as $c^2 \neq 1$, which are branch points. If we know u_0 (the function value at $v = 0$) and u_1 (the derivative value), we can compute u_2 directly from the differential equation; from there on the recurrence relation gives the Taylor coefficients.

If the degree of the polynomials in the coefficients is larger, then the order of the recurrence relation is likewise larger. This means that special code must be generated to start the recurrence relation. For instance, if we instead transformed more like Mathieu originally did, with $x = \arccos(c + v)$ (without the factor $1/2$), then we get a fourth-order recurrence relation because the function $\cos 2x$ transforms to a degree 2 polynomial, namely the 2nd degree Chebyshev polynomial, instead of to a linear polynomial. In that case, we not only need u_2, which we can get from the differential equation, but we also need u_3. Explicitly written out for that case,

$$u_3 = \frac{1}{6(c^2-1)^2}\left(2\left(c\left(2c^2+1\right)q - 3ac\right)u_0 + \left(2\left(c^2-1\right)\left(1-2c^2\right)q + a\,c^2 + 2c^2 - a + 1\right)u_1\right). \tag{24}$$

This formula was automatically generated in Maple by letting `diffeqtorec` know about the initial conditions $y(c) = u_0$ and $y'(c) = u_1$. The output was tidied up manually for inclusion in this paper.

The only important point about this section is that the recurrence relations for the Taylor series coefficients of the unknown solution to the differential equation about an arbitrary point can be generated automatically. We, therefore, regard this as a "solved problem" for our purposes,[1].

3.2 D-finite or holonomic functions

D-finite functions, also called holonomic functions, have been attracting significant interest in the computer algebra community for about two decades now. Many elementary and special functions are D-finite functions: the exponential function of course, sine and cosine, Bessel functions, hypergeometric functions, and the Airy functions are D-finite. Neither the tangent function nor the secant function are D-finite (although they are ratios of such) and of course, the Gamma function is not, so this classification does have some important exclusions.

One key property of D-finite functions is that because the recurrence relation for their Taylor coefficients is of fixed finite order, the Taylor coefficients can be computed quickly about any point where we know enough information to get the recurrence relation started. In ordinary floating-point arithmetic such as IEEE-754 double precision, this may not be as useful as it seems because some recurrence relations can be unstable. But in arbitrary-precision environments, many of the milder instabilities can be

[1] Still not *easy* though, and I have a lot of respect for people who can write code that does this.

handled more easily than one might think, merely by adding a few guard digits to the computation. For instance, if we are working to one hundred digits of precision, adding ten more to deal with the instability is only a small marginal cost.

The presence of nearby singularities, of course, introduces the potential for exponentially-growing instabilities in the recurrence relation. To manage this requires the use of a small stepsize.

Another important feature of the solution to these recurrence relations is that the Taylor coefficients might initially grow very large before they eventually decay, which they must do if the sequence of partial sums is to be convergent. This phenomenon, typically called "the hump" or something similar, also requires the use of guard digits to allow large steps to be taken in the solution of the differential equation. This is true for all functions, not just D-finite functions.

Some functions, such as the Mathieu function, are not themselves D-finite because their defining differential equation does not have polynomial coefficients. However, by a change of variable, one can find a D-finite formulation, as I showed above and which Mathieu himself used in order to simplify hand computation of the Taylor coefficients. On the other hand, it is not quite as simple as it might seem. The transformations $v = \cos(x)$ and $u = \sin(x)$ which Mathieu used have places in their range where dv/dx is zero; indeed the first one has such a point right at $x = 0$ which is where the initial conditions for the Mathieu equation are given. These flat spots mean that the change of variable is not locally invertible. The transformations $v = \sin(x/A)$ for integer $A \geq 2$ at least make the first flat spot appear for $x \geq 2\pi$, ensuring that the important interval $0 \leq x < 2\pi$ has no such spots; but the larger A is, the higher the degree of $\cos 2A \arcsin(v)$ (these are Chebyshev polynomials, up to sign). And this means that the recurrence relation is longer, and more work has to be done to get the recurrence relations started.

There are solutions to this—for instance, one could use more than one D-finite formulation and "patch" them together—but it's not straightforward. Indeed for the papers [8, 9], we simply used the $O(m^2)$ cost Cauchy products and the original differential equation, because we were not interested in the method *per se* but rather in the accurate computation of Mathieu functions and generalizations in order to solve the hemodynamics problem.

3.3 Implicitness

"Some degree of implicitness seems to be necessary."

— C. William "Bill" Gear, at a 1991 conference in Los Alamos

Knowing the initial conditions, one can expand the solution in Taylor series at the initial point. To take a step, one could use an explicit Taylor series method; however, for the Mathieu equation, which is (highly) oscillatory even if not actually stiff [45], this is not completely satisfactory. I chose instead to use an *implicit* method, which also seems natural because one uses the Taylor coefficients at both ends of the step, which increases accuracy and might be useful even for nonstiff problems. This requires expanding the solution about the tentative next knot.

For a d-dimensional system or a dth-order equation, this means computing d linearly independent solutions at that point, and later we must find some method of identifying the constants combining those independent solution to give us what we want. That is, we want to compute the fundamental solution $\mathbf{Y}(x)$ that satisfies $\mathbf{Y}(x_{n+1}) = \mathbf{I}_d$, the d by d identity matrix.

Once we have used these d linearly independent solutions to identify the Taylor series at the next knot, and the step has been accepted, then we have the desired Taylor series for the left half of the new blend needed for the next step; this is analogous to the economy of FSAL (First Same as Last) in some Runge–Kutta methods. This means that solving a dth-order equation by this implicit method requires d Taylor series generations per step; if the problem is D-finite then the cost will be $O(dm)$ flops, while for general problems the cost will be $O(dm^2)$. One therefore wants to be able to take a timestep at least d times larger using this implicit method than would be permitted by an explicit Taylor series method.[2]

For a simple scalar 2nd-order equation like the Mathieu equation, I chose to compute the two independent solutions satisfying first $y(x_{n+1}) = 1$ and $y'(x_{n+1}) = 0$ and second $y(x_{n+1}) = 0$ and $y'(x_{n+1}) = 1$.

Once we have Taylor coefficients for $y(x)$ at $x = x_n$ and an approximation to the fundamental solution $\mathbf{Y}(x)$, we may conceptually form the blend $H(x)$ that matches the Taylor coefficients at $x = x_n$ and $\mathbf{Y}(x)\mathbf{a}$ at $x = x_{n+1}$ with as-yet unknown coefficients $\mathbf{a}$. We now wish to find a method to set up equations to solve for the unknowns $\mathbf{a}$.

3.4 Collocation for 2nd-order equations and higher

Because one main application of the method in this paper is the computation of *functions* that are the solution of second-order equations, and because the sensitivity of such functions is frequently itself of interest, I decided to implement direct control of the residual (defect) instead of local error control as is more usual and to estimate the forward error by using Green's functions. This is effective because accurate approximations to integrals (and hence Green's functions) of the solution are directly available by the exact quadrature of the blendstring used to represent the solution.

The residual $r(x)$ is defined for the differential equation $a_2(x)y'' + a_1(x)y' + a_0(x)y = 0$ by simply substituting the computed solution $z(x)$ back into the differential equation:

$$r(x) := a_2(x)z''(x) + a_1(x)z'(x) + a_0(x)z(x)\ . \tag{25}$$

The derivatives of the blendstring representing $z(x)$ are available by (semi)-automatic differentiation, and are exact derivatives of the blendstring, up to rounding errors, which are a small multiple of the unit roundoff μ, which itself is controlled by setting the precision at which we are working.

[2] In my experiments this always occurred, and the savings are usually dramatic. Even in just ordinary double precision, a typical result is that the Taylor series method might take (say) 15 steps whereas the Hermite–Obreshkov method takes just 4, using the same grade $m = 22$ Taylor series at each end. Moreover, the Hermite–Obreshkov method is usually orders of magnitude more accurate, even with that much lower cost. However, a detailed study is needed because the work-precision relationship is not really captured by the $O(h^m)$ versus $O(h^{2m})$ asymptotic error estimates.

To take a step from $x = x_k$ to $x = x_{k+1}$, we set up a blend $L(x)$ which has all its Taylor coefficients given by the known Taylor coefficients at the left end, blended with all zero Taylor coefficients at the right endpoint. This is a polynomial of grade $2m + 1$ and is not simply the Taylor polynomial at the left; its values and derivatives at x_{k+1} are all zero. We next blend zero coefficients at x_k with the Taylor series of each of the d linearly independent computed solutions $\mathbf{Y}_j(x)$ at x_{k+1}. We write $y(x) = L(x) + \sum_{j=1}^{d} c_j \mathbf{Y}_j(x)$ and seek to determine the unknown coefficients c_j.

We set the residual to be zero at d collocation points, namely the Chebyshev–Lobatto points, in the interval (x_k, x_{k+1}). This gives us d linear equations in the d unknowns. For the Mathieu equation, $d = 2$ and the linear system was observed to be well-conditioned in practice. An analysis along the lines of [2] needs to be carried out to determine under what conditions this happens in general.

To be precise, denote the d collocation points by $x_{k,i} = x_k + \eta_i(x_{k+1} - x_k)$ with $0 < \eta_1 < \eta_2 < \cdots < \eta_d < 1$, and let $V_{i,j} = a_2(x_{k,i})\mathbf{Y}_j''(x_{k,i}) + a_1(x_{k,i})\mathbf{Y}_j'(x_{k,i}) + a_0(x_{k,i})\mathbf{Y}_j(x_{k,i})$. Here $\mathbf{Y}_j(x)$ is the jth fundamental solution satisfying $\mathbf{Y}_j(1) = \mathbf{e}_j$. Set the vector $\mathbf{b}$ to be

$$\mathbf{b} = -\begin{bmatrix} a_2(x_{k,1})L''(x_{k,1}) + a_1(x_{k,1})L'(x_{k,1}) + a_0(x_{k,1})L(x_{k,1}) \\ a_2(x_{k,2})L''(x_{k,2}) + a_1(x_{k,2})L'(x_{k,2}) + a_0(x_{k,2})L(x_{k,2}) \\ \vdots \\ a_2(x_{k,d})L''(x_{k,d}) + a_1(x_{k,d})L'(x_{k,d}) + a_0(x_{k,d})L(x_{k,d}) \end{bmatrix}. \tag{26}$$

Then, solving the linear system $\mathbf{Vc} = \mathbf{b}$ will identify the coefficients in the approximate solution $y(x)L(x) + \sum_{j=1}^{d} c_j \mathbf{Y}_j(x)$ on $x_k \le x \le x_{k+1}$.

Thus each step requires the computation of d sets of Taylor coefficients at the tentative knot x_{n+1} together with the evaluation of the residual (which requires d derivatives of the $d + 1$ blends) and finally the solution of a $d \times d$ linear system of equations. This cost is higher than simply using explicit Taylor series, but as could have been expected for stiff or oscillatory systems this extra cost pays off with the ability to take considerably larger stepsizes.

For nonlinear problems, one may use quasilinearization as usual. Again, as usual, this requires a good initial estimate of the solution.

3.5 Defect control (Residual control)

I wanted to control the residual in the original second-order formulation of the Mathieu equation, instead of using the more usual mathematically equivalent first-order system. The boundary-value problem code COLSYS is the only other solver that I have used which does this [3]. I have several reasons for doing it this way, which I hope to discuss in detail in a future paper. For now, I'll just give two reasons. First, doing it this way instead of reformulating to a first-order system allows direct use of the Green's function and makes analysis of the quality of the solution easier. Second, one can give a direct computation of the *optimal* backward error in this context, not just the backward error of the computed interpolant. This idea has some interesting subtleties; for the beginning of this analysis, see [16, 17].

As a practical matter, to keep the residual small I imposed (for a second-order equation) $r(x) = 0$ at two collocation points on the interval. For approximation theoretic reasons, I chose the Chebyshev–Lobatto points $x_n + h/4$ and $x_n + 3h/4$. For higher-order systems, one would choose the higher-order Chebyshev–Lobatto points for similar reasons.[3] Since the Hermite–Obreshkov method that I use uses Taylor approximation of grade m at each end of the interval, this ensures automatically that $r(x)$ and all its derivatives up to the $(m-1)$st are zero at either end of the interval. This means that the form of the leading term of the residual error is (in terms of the unit interval variable s)

$$r(s) = Ks^{m-1}(s - 1/4)(s - 3/4)(1 - s)^{m-1} . \tag{27}$$

This is $O(h^{2m})$ in the x variable. As is usual, equidistribution of the error ensures that the error for the whole integration will be $O(\overline{h}^{2m})$ where $\overline{h}$ is the arithmetic mean stepsize taken [12]. See also [13, 32]. Use of $m = 20$ is not uncommon, so the method would then be of 40th order. At this high an order, the method behaves more like a spectral method, and the number of steps taken is usually quite small, sometimes only two or three across $[0, 2\pi]$. This means that what is making the error small is not the order *per se* because even $\overline{h}$ might be larger than 1, but rather the smallness of the error coefficients. I have used the code with m as high as 100, but usually, the efficiency returns are diminishing by that high an order.

For the Mathieu equation, I chose to solve for both linearly independent solutions at once, and imposed the error control on both components at once; I measured the residual at the maximum of the polynomial, namely $s = 1/2$, and relied on the fact that the error coefficient K—which is different for each component—cannot be zero for both components at once. This gave quite a reliable test for the maximum residual on the step, albeit one that came at a slightly increased cost per step, namely the cost of computing one extra pair of residuals per step, and the cost of solving one more linear system. These costs are normally shadowed completely by the cost of generating $d = 2$ sets of Taylor coefficients at the putative next knot.

An alternative would be to measure the residual at more than just one point. Doing so is cheap because the Taylor coefficients do not need to be recomputed: all that needs to be done is to evaluate $y(x)$ and its derivatives at more points. In practice, I did this afterwards, to give an *a posteriori* reassurance that the residual was everywhere small. Indeed I frequently evaluated the residual at hundreds of points in each subinterval, which was overkill.

For a general residual control strategies that are more efficient than this in a Runge–Kutta setting, see [22, 23].

I chose the smooth error control heuristics of [27] in order to use the information provided by sampling the maximum residual to control the stepsize of the integration. These heuristics require a starting method, and I used an extrapolated estimate from a simple Taylor series method for the first step. To be specific, I chose an initial stepsize

[3] For odd-order equations, even for just third-order equations, this introduces a difficulty: the location of the maximum of the residual no longer occurs at $s = 1/2$ and indeed the first term of the residual error is zero there. For even-order equations, this is not a problem.

$h_1 = x_1 - x_0$ on the first interval for which the Taylor coefficients at $x = x_0$ predicted an error small enough to satisfy the tolerance but extrapolated so that Kh_1^{2m} was equal to the tolerance, not Kh_1^m. This seemed to work well in practice, though more testing and refinement of the heuristic may be necessary to make it work in general.

3.6 Forward error and Green's functions

The usual connection made between the residual $r(x)$ and the forward error $y(x)-z(x)$ in the numerical solution $z(x)$ of (nonlinear) IVP for ODE uses the Gröbner–Alexeev nonlinear variation-of-constants formula

$$y(x) - z(x) = \int_{\xi=0}^{x} G(x,\xi) r(\xi)\, d\xi \ . \tag{28}$$

Here, $y(x)$ is the reference solution of the original equation and we assume that no further errors are introduced in the initial conditions. See [28] for more details.

But for second-order linear problems $a(x)y''(x) + b(x)y'(x) + c(x)y(x) = r(x)$, the theory of Green's functions is all we need, exactly as it is taught in undergraduate courses in ODE. By finding two independent solutions $y_1(x)$ and $y_2(x)$ and computing the Wronskian[4] $W(x) = y_1(x)y_2'(x) - y_1'(x)y_2(x)$ one can explicitly construct the Green's function

$$G(x,\xi) = \frac{y_1(\xi)y_2(x) - y_1(x)y_2(\xi)}{a(\xi)W(\xi)} \tag{29}$$

Then, an expression for the forward error is

$$z(x) - y(x) = \int_{\xi=0}^{x} G(x,\xi) r(\xi)\, d\xi \ . \tag{30}$$

Notice that if we have blendstrings for $y_1(x)$ and $y_2(x)$ and if (as is true for the Mathieu equation) $a(x) = 1$ and the Wronskian is constant, then we may explicitly construct the Green's function as a blendstring, because blendstrings can be explicitly integrated, generating another blendstring. Once constructed, it can be plotted and bounded or otherwise used to understand the connection between backward and forward errors.

4 Not A-stable, but decently stable

Consider the *oscillatory test problem*[5]

$$\ddot{y} + \omega^2 y = 0 \tag{31}$$

[4] By using Abel's identity $W' = -b(x)W(x)/a(x)$ we could construct the Wronskian even if we didn't know the independent solutions; this generalizes to higher-order linear equations as well. See any ODE textbook, but the Wikipedia article on the subject is both correct and convenient.

[5] Some of the material in this section appeared in abbreviated form in [15]. The analysis is along standard lines. I include these details for comprehensibility and convenience.

with initial conditions $y(0) = y_0$ and $\dot{y}(0) = y_1$. Here the dot $\cdot$ means differentiation with respect to t, which we think of as time. This has the reference solution $y(t) = y_0 \cos(\omega t) + y_1 \sin(\omega t)/\omega$. The frequency parameter ω can be absorbed into the time variable t but it is worthwhile to leave it explicitly present for the moment.

Now, taking a single step of size h with the exact solution finds the exact solution value Y_0 and derivative value Y_1 at $t = h$, with the vector of initial conditions being multiplied by a certain matrix, which I will call $\mathbf{A}$.

$$\begin{bmatrix} Y_0 \\ Y_1 \end{bmatrix} = \begin{bmatrix} \cos \omega h & \frac{\sin \omega h}{\omega} \\ -\omega \sin \omega h & \cos \omega h \end{bmatrix} \begin{bmatrix} y_0 \\ y_1 \end{bmatrix} . \tag{32}$$

This can be written more intelligibly as

$$\begin{bmatrix} Y_0 \\ Y_1/\omega \end{bmatrix} = \begin{bmatrix} \cos \nu & \sin \nu \\ -\sin \nu & \cos \nu \end{bmatrix} \begin{bmatrix} y_0 \\ y_1/\omega \end{bmatrix} . \tag{33}$$

but this will make no difference to the analysis below.

Because the equation is autonomous, this step from $t = 0$ to $t = h$ is exactly the same as the same width step from $t = t_k$ to $t = t_k + h$. Timesteps with this exact solution therefore satisfy $\mathbf{y}^{(k)} = \mathbf{A}^k \mathbf{y}^{(0)}$. The eigenvalues of this matrix satisfy the characteristic equation $\lambda^2 - 2 \cos \omega h \lambda + 1 = 0$ and are $\exp(\pm i \omega h)$, which both have magnitude 1, implying that the length of the vector of initial conditions does not grow or decay exponentially. One would like this property to hold with the numerical method, if possible. Applying the collocation method just described gives, at every balanced order (grade m Taylor polynomials at each end), an analogous matrix, but with rational functions $C_m(\nu)$ and $S_m(\nu)$ and $\hat{S}_m(\nu)$ (easily computed[6] for any fixed m) of $\nu = \omega h$ in place of $\cos \nu$ and $\sin \nu$:

$$\mathbf{A}_m := \begin{bmatrix} C_m(\nu) & \frac{\hat{S}_m(\nu)}{\omega} \\ -\omega S_m(\nu) & C_m(\nu) \end{bmatrix} . \tag{34}$$

Because it turned out (at least for all $m \leq 100$, which I computed) that the determinant $C_m^2 + S_m \hat{S}_m = 1$ exactly, the matrix has characteristic polynomial $\lambda^2 - 2C_m(\nu)\lambda + 1$, implying that the product of its two eigenvalues is 1 and the map is thus exactly area-preserving. The eigenvalues are $C_m(\nu) \pm i\sqrt{1 - C_m^2(\nu)}$. However, the eigenvalues *both* have magnitude 1 if and only if $|C_m(\nu)| \leq 1$.

Note that both h and ω, hence $\nu = \omega h$, are real. This suggests investigating the real zeros of the equation $C_m^2(\nu) - 1 = 0$. The first few rational functions $C_m(\nu)$ are tabulated in Table 1. They are some kind of rational approximation to $\cos \nu$, but I do not recognize them (they are not (m, m) Padé approximants, for instance).

Once the value of $C_m(\nu)$ becomes larger than 1 in magnitude, the eigenvalues of $\mathbf{A}_m$ are no longer of unit modulus, and $\lambda_1 = C_m(\nu) + \text{signum}(C_m(\nu))\sqrt{C_m(\nu)^2 - 1}$ will be larger than 1 in modulus and therefore the lengths of the vectors $[y_k, y_k']$ will

[6] But nonetheless I made a blunder in [15], in thinking that $S_m(\nu)$ and $\hat{S}_m(\nu)$ were the same. They are not. Luckily that blunder was of no consequence because we only need to work with $C_m(\nu)$.

Table 1 Collocation at Chebyshev–Lobatto points: The first few rational approximations to cosine and the first positive zero of $C_m^2 - 1$ as a fraction of π. The next entry is too wide for this table but has $\nu^*/\pi \approx 1 + 1.25 \times 10^{-10}$. The sequence of the next larger zeros is, starting at $m = 1$: $\hat{\nu}/\pi = 1 + 0.0396$, $1 + 0.0119$, $1 + 0.00110$, $1 + 5.11 \times 10^{-5}$, $1 + 1.42 \times 10^{-6}$, and $1 + 2.65 \times 10^{-8}$. See the worksheet https://github.com/rcorless/Blends-in-Maple/blob/main/collocationanalysisHOmethod.mw

m	$C_m(\nu)$	ν^*/π
1	$\frac{57\nu^4-1408\nu^2+3072}{9\nu^4+128\nu^2+3072}$	0.94035
2	$-\frac{2\left(11\nu^6-1353\nu^4+28160\nu^2-61440\right)}{3\nu^6+146\nu^4+5120\nu^2+122880}$	0.99817
3	$\frac{25\nu^8-9016\nu^6+676560\nu^4-12072960\nu^2+25804800}{3\nu^8+304\nu^6+16080\nu^4+829440\nu^2+25804800}$	0.99997
4	$-\frac{4\left(21\nu^{10}-17215\nu^8+3251520\nu^6-189560700\nu^4+3083673600\nu^2-6502809600\right)}{9\nu^{10}+1550\nu^8+117240\nu^6+9903600\nu^4+670924800\nu^2+26011238400}$	$1 - 9.99 \times 10^{-8}$
5	$\frac{31\nu^{12}-49560\nu^{10}+19397392\nu^8-2662020480\nu^6+133864738560\nu^4-2053804032000\nu^2+4291854336000}{3\nu^{12}+784\nu^{10}+80752\nu^8+9945600\nu^6+1099042560\nu^4+92123136000\nu^2+4291854336000}$	$1 + 5.03 \times 10^{-9}$

start to grow exponentially, like λ_1^k. This is a numerical instability of the method. To ensure that this does not happen, one must take $\boldsymbol{\nu} < \boldsymbol{\nu}^*$, or (approximately) $h < \pi/\boldsymbol{\omega}$. For high frequencies $\boldsymbol{\omega}$ one would thus seem to have to take very small timesteps.

So it would seem that there is a stability restriction $h < \pi/\boldsymbol{\omega}$ akin to the stepsize restrictions on explicit methods for stiff problems [7, 45].

But this is not the complete story, here, and the situation is better than it seems at first: $C_m^2 - 1$ has *another* zero very nearby: for $m = 3$, at 1.0011π. The maximum value that $C_m^2 - 1$ attains, on the tiny interval it is positive, is less than $3.13 \cdot 10^{-6}$. The magnitude of the largest eigenvalue is thus $1 + O(10^{-3})$. This does cause growth but, while it is technically exponential, it would not be visible in the numerical solution of the simple harmonic oscillator unless on the order of a thousand steps[7] were taken!

For larger m, the maximum λ_1 is even smaller. For the simple harmonic oscillator at least, this method is actually quite stable. There are other zeros, near 2π and 3π and so on, for larger m, and the maxima on the small positive intervals get larger and larger until the method actually fails for large enough $\boldsymbol{\nu}$, no matter how large one takes m. This is because the method is not A-stable [45], of course. But, A-stability is not wholly appropriate for oscillatory problems, and the current analysis gives more information.

For instance, with $m = 20$, the first interval is from $1 - 4.29 \cdot 10^{-45}$ to $1 + 3.65 \cdot 10^{-42}$ and the next is $2 + 3.2 \cdot 10^{-32}$ to $2 + 6.8 \cdot 10^{-32}$, with intervals growing larger and larger up to $10 + 1.6 \cdot 10^{-5}$ to $10 + 9.2 \cdot 10^{-4}$. Thus for about a hundred steps of the method, one should be able to take any $h < 10\pi/\boldsymbol{\omega}$; this *is* a stability restriction,[8] but is ten times better than the technically correct one of $h < (1 - 4.29 \cdot 10^{-45})\pi/\boldsymbol{\omega}$. And, in any case, one will want stepsizes small enough to resolve the actual oscillations.

If instead, we had chosen to reformulate the Mathieu equation as a first-order system, it would have been possible to construct an A-stable method, not using collocation. But even if we choose collocation at $s = 1/2$ in a symmetric fashion, this method is not A-stable.

Leading error coefficient The leading term of the residual is

$$E_m = y_0 \boldsymbol{\omega}^2 \times \frac{1}{(2m)!} s^{m-1}(s - \frac{1}{4})(s - \frac{3}{4})(1 - s)^{m-1} \boldsymbol{\omega}^{2m} h^{2m} + O(h^{2m+1}) \,. \quad (35)$$

Because in the code we take steps to avoid the difficulty if $y_0 = 0$, we ignore the issue here. Since one term in the equation is of the form $\boldsymbol{\omega}^2 y$ we regard the term $y_0 \boldsymbol{\omega}^2$ as a proper scaling. The error will thus be small for large m because we are dividing by $(2m)!$ and because the maximum value of the polynomial in s occurs at $s = 1/2$ and is thus $2^{-(2m+2)}$. If $\boldsymbol{\nu} = \boldsymbol{\omega} h$ is as large as π then the size of the $\boldsymbol{\nu}^{2m}$ factor can degrade the accuracy considerably. This makes the extremely weak stability restriction above even less important.

[7] Another blunder in [15] was not to convert back from the maximum of $C_m^2 - 1$ to the maximum of λ_1, which takes a square root. Again (luckily) this blunder was of little consequence.

[8] A referee points out that this is comparable to the stability restriction of the Strang splitting, an explicit method that has the stability restriction $h < 2/\boldsymbol{\omega}$. Since the Hermite–Obreshkov method is implicit, we would expect better behavior, and indeed it is by at least a factor $\pi/2$ and in practice more.

For instance, the first term in the residual series expansion when $m = 20$ is $y_0\boldsymbol{\omega}^2$ times

$$\boldsymbol{\nu}^{40}\frac{s^{19}(s-1/4)(s-3/4)(1-s)^{19}}{40!}. \tag{36}$$

Keeping this term of the residual smaller than ε with $m = 20$ requires $\boldsymbol{\nu} < (2^{42}\cdot 40!\varepsilon)^{1/40} \approx 32\varepsilon^{1/40}$; if ε is about 10^{-40} (otherwise such a high-order method is not really justified) this gives $\boldsymbol{\nu} < 3.2$, roughly the same restriction on h as the strictest stability restriction above. When we increase $\boldsymbol{\nu}$ by allowing λ_1 to be as large as, say, $1 + O(10^{-3})$ at this order, we can have $\boldsymbol{\nu}$ as large as 9π, so $\boldsymbol{\nu} < 30$ approximately. But at this size, the leading term of the residual error is $O(1)$, and so the computation would likely be worthless anyway.

We therefore conclude that while this method can be unstable by a strict accounting, the instability is of no real importance.

Collocation at other points

Collocation at the Chebyshev–Lobatto points is not the only choice. One might collocate instead at $s = 1/2$, and get two equations by insisting not only that the residual should be zero but also the derivative of the residual. This is simple enough in practice. However, if one looks closely at the results in Table 2, one sees that the stability behavior is very similar, perhaps slightly worse. The error coefficients (not shown here) are a factor of m smaller, which is an advantage, but the location of the maxima are at $(1 \pm m^{-1/2})/2$, which makes them narrow (given the location of the zero at $1/2$) and possibly awkward to sample at. Choosing instead the points $s = 0$ and $s = 1$, which would require generating one more term in the Taylor series at each end, gives an error about 4 times worse than the Chebyshev–Lobatto points. This makes the choice of Chebyshev–Lobatto points reasonable.

As previously noted, there are difficulties that arise for odd-order equations, even just for third order, in that the location of the maximum residual is no longer just at $s = 1/2$. I believe that these difficulties can be overcome, just as they have been overcome for defect-controlled Runge–Kutta methods [22].

5 Experimental results

In this section, I will report several experiments using my Maple code to solve the Mathieu equation and the modified Mathieu equation, at various precisions. I point out that Maple is an *interpreted* language, with a compiled kernel and some features

Table 2 Collocation at $s = 1/2$: The first few rational approximations to cosine and the first positive zero of $C_m^2 - 1$ as a fraction of π. The next entry is too wide for this table but has $\boldsymbol{\nu}^*/\pi = 0.99997$

m	$C_m(\boldsymbol{\nu})$	$\boldsymbol{\nu}^*/\pi$
1	$\frac{5\boldsymbol{\nu}^4-96\boldsymbol{\nu}^2+192}{\boldsymbol{\nu}^4+192}$	0.90032
2	$-\frac{2\left(3\boldsymbol{\nu}^6-291\boldsymbol{\nu}^4+5376\boldsymbol{\nu}^2-11520\right)}{\boldsymbol{\nu}^6+6\boldsymbol{\nu}^4+768\boldsymbol{\nu}^2+23040}$	0.98625
3	$\frac{21\boldsymbol{\nu}^8-6072\boldsymbol{\nu}^6+403920\boldsymbol{\nu}^4-6842880\boldsymbol{\nu}^2+14515200}{3\boldsymbol{\nu}^8+48\boldsymbol{\nu}^6+6480\boldsymbol{\nu}^4+414720\boldsymbol{\nu}^2+14515200}$	0.99916

for relatively rapid evaluation of some floating-point tasks in double precision. For numerical linear algebra and the numerical solution of differential equations [43] it's comparable in speed to Matlab [42], *when working in double precision* with standard methods. Execution in a fast language like C, Fortran, or Julia can be a hundred or even a thousand times faster [38].

When working in software floating-point, even in just 20 decimal digits, the code is vastly slower.[9] Computing times at 100 decimal digits of precision are simply not comparable to those at hardware precision. Finally, my code has not been optimized for speed at all; the first and most important of those would be to rewrite it in a performant language. The second would be to use the D-finite formulation for the Mathieu equation. So I am not going to report computing times for any of these experiments. My code is available, though, so you can see for yourself that in spite of my remarks above, it's not too bad. One can get an accurate and reliable solution in "reasonable time," meaning that you usually won't get too impatient waiting for the solution. The Mathieu code is available at https://github.com/rcorless/Puiseux-series-Mathieu-double-points in the file `MathieuCodeDemo.maple`.

What this code really does is give "proof of concept." Given the experience with my laboratory code, it seems quite likely that robust and efficient code could be written that uses this method.

5.1 A typical test

We first have a look at what the code does at low precision (IEEE double precision, corresponding to `Digits:= 15` in Maple). We choose $m = 10$, so we will be using an $O(h^{20})$ method. We use a tolerance of 10^{-14} for these integrations, to start with.

We look at solving the Mathieu equation $y'' + (a - 2q\cos 2x)y = 0$ and the modified Mathieu equation $y'' - (a - 2q\cosh 2x)y = 0$, which are related by a rotation in the complex plane so I can use the same code to solve either equation merely by choosing a different path in the complex plane. For this first test, we choose

$$q = 1.4687686137851419923072930898619635684 77i \tag{37}$$

(reporting this here to 32 Digits even though we'll only use 15 in this test) and

$$a = 2.0886989027496954074221070500473124906 59\,. \tag{38}$$

[9] One referee pointed out a speed comparison among competing software systems that claimed an advantage for one particular package. I do not reproduce that link here because I do not think the comparison there was fair. Maple uses MPFR [25] and GMP https://gmplib.org/ and is quite competitive at high precision, given that it works in a variable precision environment and precision can change at any moment, which makes memory usage unpredictable. It has been known for a long time that fixing the (high) precision ahead of time allows much more rapid computation; see, e.g., [4]. But to do that, one should not use a Problem Solving Environment like Maple or Matlab, but something designed with efficiency instead of convenience as the prime consideration. See David Bailey's web page https://www.davidhbailey.com/dhbsoftware/ for Fortran code and see also Fredrik Johansson's web page https://fredrikj.net/ for a description of his packages FLINT and Arb; note that as of 2022 Arb is now available in Maple.

These are the parameter q and eigenvalue a values for the Mulholland–Goldstein double eigenvalue; no other existing Mathieu function code that I know of is set up to cope with this double eigenvalue.

We compute the Mathieu function $\mathrm{ce}_0(x;q)$ on $0 \le x \le \pi$, which satisfies $\mathrm{ce}_0(0;q) = 1$ and has flat slope at $x = 0$, and simultaneously compute the second solution of the Mathieu equation which has $y(0) = 0$ and $y'(0) = 1$. Both of these functions are complex-valued.

We then increase the precision to 30 decimal digits so that the residual can be accurately computed for $\mathrm{ce}_0(x;q)$. When we plot the absolute value of the residual, see Fig. 2, we see that the integration at this tolerance took eight timesteps to cross the interval $[0, \pi]$, and that the measured residual is maximal near the midpoint of each step, and that the maximum residual is smaller than the tolerance. We also see that after the initial step, the residual decreases as the integration proceeds, which suggests that the constants in the control heuristic are slightly too conservative; we could perhaps save a timestep by tweaking them. That the residual on the initial step is too small suggests that heuristic, also, can be improved. There is a smaller residual on the final step because the end of the interval occurs some time inside the final step.

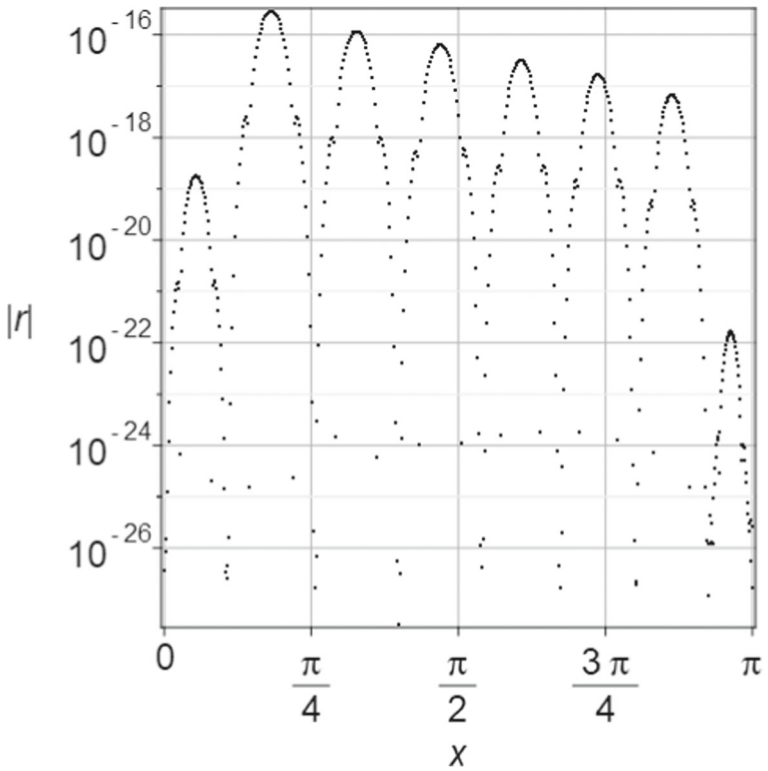

Fig. 2 The absolute value of the residual $|y'' + (a - 2q\cos x)y|$ for the solution computed using double precision, $m = 10$ and tolerance 10^{-14}. Residual computed in 30 digit precision

This graph is quite typical. I have performed many residual tests like this (perhaps hundreds) and once the code was debugged the graphs all looked like this in general, differing only in the particulars such as the number of time steps.

If we change $m = 10$ to $m = 15$ and repeat the computation, we find that the integration takes only four timesteps to cross the interval $[0, \pi]$ (and takes about half the CPU time, but I said I wasn't going to report on that).

If we instead decrease m to $m = 5$ (so the method is now only tenth order) and repeat the computation, we find that the integration takes 65 steps and considerably more time (although each step is cheaper). Again, however, the measured residual is always less than 10^{-16} and decreases as the integration proceeds.

If instead, we increase the precision of the computation to 30 decimal digits, and the precision of the computation of the residual to 60 digits, and use a tolerance of 10^{-29} and $m = 15$, we find that the code takes 13 steps to cross the interval $[0, \pi]$. It does so in less time than the double-precision code with $m = 5$ does. The residual is uniformly less than 10^{-31} and again decreases as the integration proceeds, to a maximum of 10^{-35} on the penultimate step.

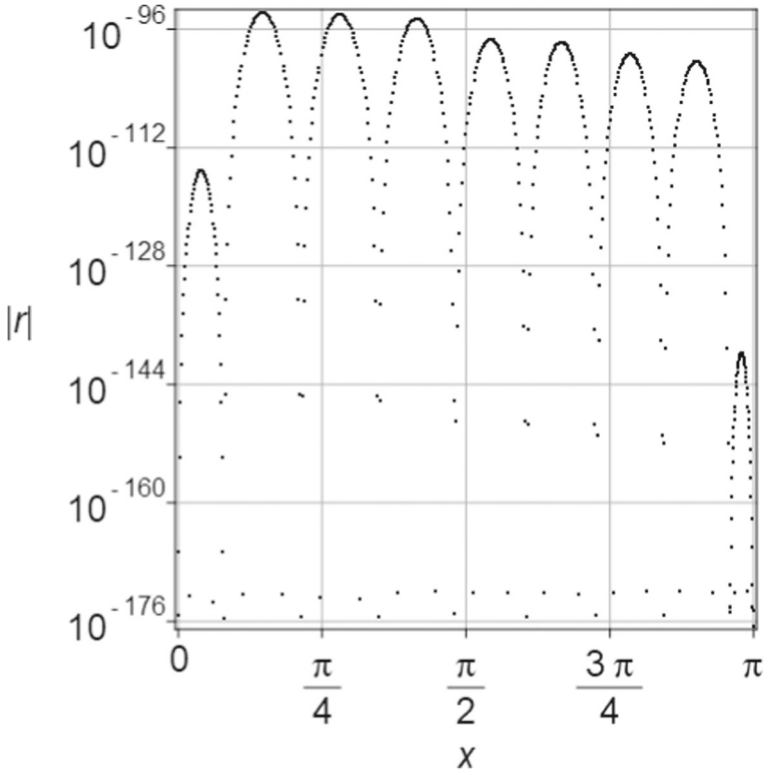

Fig. 3 The residual (computed at twice 90 decimal digits) for a 90th order method applied to the Mathieu equation using 90 digit precision during the computation, in honor of John Butcher's 90th birthday. We see that the method takes nine steps, one per decade, to solve the problem, and everywhere has more than 95 digits of accuracy in the answer

If we now increase m to $m = 40$, so we are using an 80th-order method, the code takes just three steps to cross the interval and the residual is less than 10^{-38}. This high an order code is overkill for this precision.

In honor of John Butcher's 90th birthday, though, we ask for 90 digit precision and solve with $m = 45$, giving a 90th-order method. This takes just 9 steps, and the residual is uniformly less than 10^{-95}. This seems to be just about the right amount of effort for this much accuracy: one step per decade! See Fig. 3.

Now, we try the same thing but with integration along the straight line from the origin to $2i$. I will denote this interval by $[0, 2]i$. This gives access to the modified Mathieu functions. We see in Fig. 4 that the residual *increases* over the course of the integration (but still stays within tolerance). This is because the solution itself is not purely oscillatory, but rather is increasing. The code actually uses *relative* residual, by the way: it accepts or rejects steps depending on whether the error is small compared to the tolerance times an estimate of the magnitude of the solution. It's still somewhat conservative, though, as can be seen in the figure.

In fact, the solution increases *doubly* exponentially with x increasing along the imaginary axis. If we plot the residual not on $[0, 2]i$ but rather (after redoing the

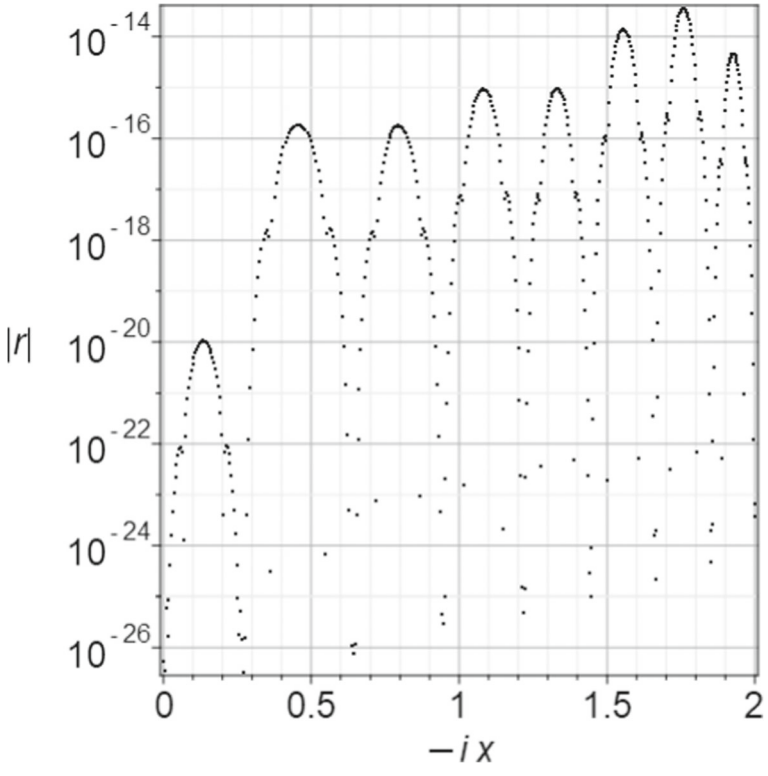

Fig. 4 The absolute value of the residual $|y'' + (a - 2q\cos x)y|$ for the solution computed using double precision, $m = 10$ and tolerance 10^{-14}. Residual computed in 30 digit precision. Unlike in Fig. 2, integration proceeds vertically up the imaginary axis to $2i$

integration) on the interval $[0, 3]i$, the magnitude of the solution at $x = 3i$ is already 10^6. By $3.5i$, the magnitude is $O(10^{10})$. By $4i$ it is $O(10^{18})$. In Fig. 5, which only goes to $x = 3i$, we see that the code is working considerably harder to ensure that the residual error stays within tolerance; instead of just 8 steps to get to $x = 2i$, it needs 25 to get to $x = 3i$. That is, more than twice as many steps to get half again as far.

The *frequency* is also increasing exponentially.

The increasing difficulty of integrating this equation accurately as x goes farther up the imaginary axis is one reason that I wrote this code in the first place. Standard codes have a hard time, although they do better than one might think. The real problem is in *verifying* how good a job they might have done. The Hermite–Obreshkov method automatically supplies high-quality interpolants with which one can compute the residual (anywhere) and verify that the solver has done a good job.

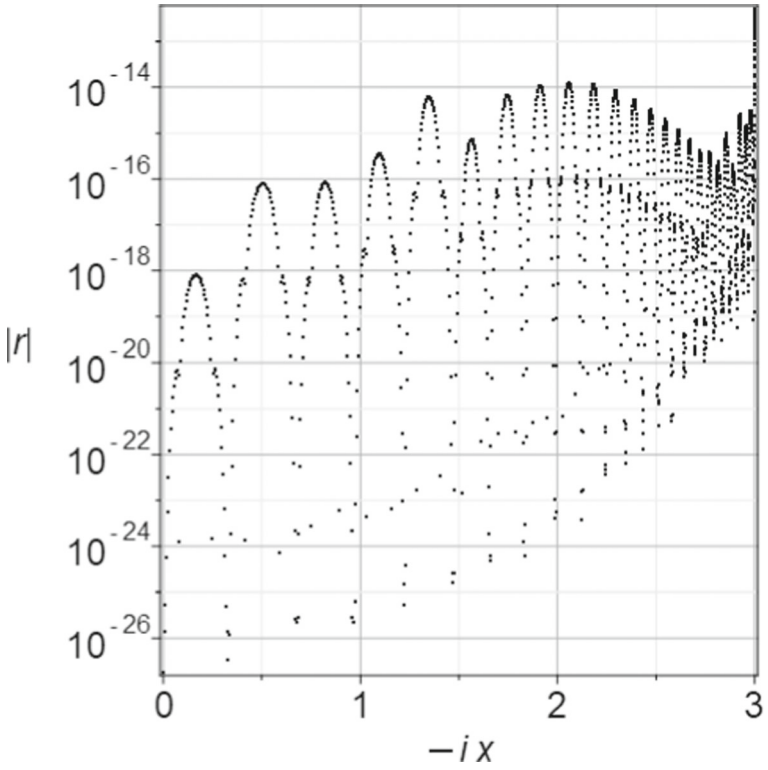

Fig. 5 The absolute value of the residual $|y'' + (a - 2q\cos x)y|$ for the solution computed using double precision, $m = 10$ and tolerance 10^{-14}. Residual computed in 30 digit precision. The integration proceeds vertically up the imaginary axis to $3i$, farther than in Fig. 4

6 Concluding remarks

The Maple prototype implementation I constructed shows that this method can be useful for the niche application of high-precision computation of mathematical functions. This method could be of particular interest for D-finite functions, for which the computation of Taylor series coefficients is particularly inexpensive in computing time.

The Mathieu functions are of some practical interest, and they are difficult to compute for certain parameter values. This method proved to be a useful way to compute them, especially for solving the Helmholtz equation in the paradoxically difficult case of nearly-circular ellipses, which is numerically difficult because the coordinate transformation that gives rise to the Mathieu functions is nearly singular there [9]. This requires computing the Mathieu equations for large imaginary values of x, which as we have seen involves doubly exponential growth and exponentially increasing frequency.

Fast and *robust* code for the Mathieu functions which includes the case of the generalized eigenfunctions needed at double eigenvalues has yet to be developed; this present code can be used by someone willing to adjust parameters such as tolerance and the grade of approximation and the number of digits of precision, but a general scientist would prefer something more bulletproof and automatic (not to mention with a better user interface).

6.1 On variable order

There are some publications that discuss heuristics for changing the order of the Taylor series as the integration proceeds, such as [5]. Variable-order methods have proved to be useful for standard solvers, especially variable-order multistep methods; one suspects that the same would be true for Taylor series-based methods, especially because the change in order is so simple: just compute one more (or one fewer) Taylor coefficient, if the heuristics indicate that higher (or lower) order would be beneficial. One thing to be aware of is that noticeably unbalanced blends have poor approximation-theoretic properties, and so one would want only gradual changes in order as the integration proceeded.

6.2 Incorporating discontinuities

A discontinuity of any sort inside the interval $0 < s < 1$ is a difficulty for this method, and there's no getting around that. If, however, one has a jump discontinuity in a value or a derivative exactly at a knot—and why *would you* not place a knot at a known discontinuity—then a blendstring can be perfectly accurate if the code is set up to have "two-sided" Taylor coefficients at such a knot. Currently, my code does not have such a feature, but I foresee no difficulty in adding it.

Extending the code so that *poles* or *branch points* can be dealt with will be more work. Hermite interpolational polynomials will not work near such points, and so the underlying interpolation scheme would have to be extended to rational or logarithmic interpolational schemes. This would certainly be worthwhile on its own.

6.3 Nonlinear problems

Nonlinear problems are typically handled by quasilinearization [3]; that is, by replacing the nonlinear ODE by a sequence of linear ODEs obtained by Fréchet differentiation about an approximate solution. This is a well-known technique with reasonably well-understood advantages and disadvantages. Preliminary experiments (not reported here) suggest that it can work well. As usual, the success or failure of the technique depends strongly on the quality of the initial approximation.

For nonlinear problems, though, explicit methods (including explicit Taylor series methods) become more competitive: the penalty of small stepsizes for mildly stiff problems can be outweighed by the cost of iteration to deal with nonlinearities. Still, I think that for some problems the Hermite–Obreshkov approach could be a good choice.

6.4 Detection and location of singularities

Linear problems tend to have singularities at known locations, but nonlinear ones can have them appear anywhere. The usual adaptive stepsize control heuristics are actually quite good at locating singularities in the path of integration (at which point the integration stops) but we have the potential for an even better approach, given that we are computing Taylor coefficients. Taylor coefficients can be used, in a technique originally due to Daniel Bernoulli but often attributed to Darboux, both to detect when singularities are near and to locate them [19]. Once located, the path of integration can be altered in a technique known as "pole vaulting" in order to avoid them.

Adding code to use the Taylor series about the tentative next step x_{n+1} to detect and locate nearby singularities would add significantly to the robustness of this code, and one potential application for this would be homotopy continuation methods for solving nonlinear algebraic equations.

One may also use Padé methods, as was done in [24]. This seems, indeed, to be the best approach.

6.5 The next step for future work

Writing a robust and efficient general-purpose solver for holonomic (D-finite) systems of arbitrary order and dimension using this method is a grand goal. Towards that end, extending the current code to handle scalar problems of arbitrary order seems a logical first step. So that is what I will try to do next.

Acknowledgements I thank my colleague Mair Zamir for getting me interested in the Mathieu equation, and Chris Brimacombe for his enthusiasm for the subject and for searching out various methods. I thank Erik Postma for teaching me some interesting Maple programming tidbits which made the code both faster and more maintainable. I thank my friends George Corliss, Ned Nedialkov, and John Pryce for teaching me about Taylor series methods and Hermite–Obreshkov methods, and for comments on an earlier draft of this paper. I thank Nick Trefethen for encouraging remarks about the theory of blendstrings. I haven't forgotten Y.F. Chang, either, and I wish him well, wherever he is. I thank Wayne Enright for teaching me about defect control, and Larry Shampine for teaching me about the importance of solution "quality." I especially thank Silvana Ilie for her proof that Taylor series methods (including Hermite–Obreshkov methods) are of cost

polynomial in the number of bits of residual accuracy requested. That result is foundational for this whole approach.
But most of all I thank John Butcher for teaching me so very many things about the numerical solution of ordinary differential equations and about interpolation of those solutions. In particular, the contour integral method has become my go-to method and makes such short work of many of the problems we encounter. In particular, it was of central use for this paper. Thank you *very* much, John, for everything.

Funding This work was partially supported by NSERC under RGPIN-2020-06438 and by the grant PID2020-113192GB-I00 (Mathematical Visualization: Foundations, Algorithms and Applications) from the Spanish MICINN.

Data availability Not applicable

Declarations

Ethical approval Not applicable

Consent to participate Not applicable

Consent for publication Not applicable

Conflict of interest The author declares no competing interests.

References

1. Abad, A., Barrio, R., Blesa, F., Rodríguez, M.: Algorithm 924: TIDES, a Taylor series integrator for differential equations. ACM Trans. Math. Softw. (TOMS) **39**(1), 1–28 (2012)
2. Ascher, U., Bader, G.: Stability of collocation at Gaussian points. SIAM J. Numer. Anal. **23**(2), 412–422 (1986)
3. Ascher, U., Christiansen, J., Russell, R.D.: COLSYS–a collocation code for boundary-value problems. In: Codes for Boundary-Value problems in ordinary differential equations, pp. 164–185. Springer (1979)
4. Bailey, D.H., Barrio, R., Borwein, J.M.: High-precision computation: mathematical physics and dynamics. Appl. Math. Comput. **218**(20), 10106–10121 (2012)
5. Barrio, R., Blesa, F., Lara, M.: VSVO formulation of the Taylor method for the numerical solution of ODEs. Comput. Math. Appl. **50**(1–2), 93–111 (2005). https://doi.org/10.1016/j.camwa.2005.02.010
6. Bornemann, F.: Accuracy and stability of computing high-order derivatives of analytic functions by Cauchy integrals. Found. Comut. Math. **11**(1), 1–63 (2010). https://doi.org/10.1007/s10208-010-9075-z
7. Bou-Rabee, N., Sanz-Serna, J.M.: Geometric integrators and the Hamiltonian Monte Carlo method. Acta Numer. **27**, 113–206 (2018)
8. Brimacombe, C., Corless, R.M., Zamir, M.: Computation and applications of Mathieu functions: a historical perspective. SIAM Rev. **63**(4), 653–720 (2021). https://doi.org/10.1137/20m135786x
9. Brimacombe, C., Corless, R.M., Zamir, M.: Elliptic cross sections in blood flow regulation. arXiv:2304.01356 (2023)
10. Butcher, J.C.: A multistep generalization of Runge-Kutta methods with four or five stages. J. ACM (JACM) **14**(1), 84–99 (1967)
11. Butcher, J.C., Hojjati, G.: Second derivative methods with RK stability. Numer. Algorithms **40**, 415–429 (2005)
12. Corless, R.M.: An elementary solution of a minimax problem arising in algorithms for automatic mesh selection. ACM SIGSAM Bull. **34**(4), 7–15 (2000)
13. Corless, R.M.: A new view of the computational complexity of IVP for ODE. Numer. Algorithms **31**, 115–124 (2002)

14. Corless, R.M.: Blends have decent numerical properties. Maple Trans. **3**(1) (2023). https://doi.org/10.5206/mt.v3i1.15890
15. Corless, R.M.: Blendstrings: an environment for computing with smooth functions. In: Proceedings of the 2023 International Symposium on Symbolic and Algebraic Computation. ACM (2023). https://doi.org/10.1145/3597066.3597117
16. Corless, R.M., Jankowski, J.E.: Variations on a theme of Euler. SIAM Rev. **58**(4), 775–792 (2016). https://doi.org/10.1137/15M1032351
17. Corless, R.M., Kaya, C.Y., Moir, R.H.: Optimal residuals and the Dahlquist test problem. Numer. Algorithms **81**(4), 1253–1274 (2019)
18. Corless, R.M., Postma, E.J.: Blends in Maple. In: Maple in Mathematics Education and Research: 4th Maple Conference, MC 2020, Waterloo, Ontario, Canada, November 2–6, 2020, Revised Selected Papers 4, pp. 167–184. Springer (2021)
19. Corliss, G., Chang, Y.: Solving ordinary differential equations using Taylor series. ACM Trans. Math. Softw. (TOMS) **8**(2), 114–144 (1982)
20. Darboux, G.: Sur les développements en série des fonctions d'une seule variable. J. Math. Pures Appl. **2**, 291–312 (1876)
21. Enright, W.H.: Second derivative multistep methods for stiff ordinary differential equations. SIAM J. Numer. Anal. **11**(2), 321–331 (1974). https://doi.org/10.1137/0711029
22. Enright, W.H., Hayes, W.B.: Robust and reliable defect control for Runge-Kutta methods. ACM Trans. Math. Softw. (TOMS) **33**(1), 1–es (2007)
23. Enright, W.H., Higham, D.J.: Parallel defect control. BIT Numer. Math. **31**(4), 647–663 (1991)
24. Fornberg, B., Weideman, J.: A numerical methodology for the Painlevé equations. J. Comput. Phys. **230**(15), 5957–5973 (2011). https://doi.org/10.1016/j.jcp.2011.04.007
25. Fousse, L., Hanrot, G., Lefèvre, V., Pélissier, P., Zimmermann, P.: MPFR: a multiple-precision binary floating-point library with correct rounding. ACM Trans. Math. Softw. **33**(2), 13–es (2007). https://doi.org/10.1145/1236463.1236468
26. Griewank, A., Juedes, D., Utke, J.: Algorithm 755: ADOL-C. ACM Trans. Math. Softw. **22**(2), 131–167 (1996). https://doi.org/10.1145/229473.229474
27. Gustafsson, K., Lundh, M., Söderlind, G.: A PI stepsize control for the numerical solution of ordinary differential equations. BIT Numer. Math. **28**(2), 270–287 (1988)
28. Hairer, E., Nørsett, S.P., Wanner, G.: Solving ordinary differential equations I. Nonstiff problems. Springer series in computational mathematics (1993)
29. Hermite, C.: Cours d'analyse de l'École polytechnique, vol. 1. Gauthier-Villars (1873)
30. Higham, N.J.: Accuracy and stability of numerical algorithms. SIAM (2002)
31. van der Hoeven, J.: Fast evaluation of holonomic functions near and in regular singularities. J. Symb. Comput. **31**(6), 717–744 (2001)
32. Ilie, S., Söderlind, G., Corless, R.M.: Adaptivity and computational complexity in the numerical solution of ODEs. J. Complex. **24**(3), 341–361 (2008)
33. Mezzarobba, M.: NumGfun: a package for numerical and analytic computation with D-finite functions. In: Proceedings of the 2010 International Symposium on Symbolic and Algebraic Computation, pp. 139–145 (2010)
34. Mezzarobba, M.: A note on the space complexity of fast D-finite function evaluation. In: Int. Workshop on Computer Algebra in Scientific Computing, pp. 212–223. Springer (2012)
35. Nedialkov, N.S., Jackson, K.R.: An interval Hermite-Obreschkoff method for computing rigorous bounds on the solution of an initial value problem for an ordinary differential equation. Reliab. Comput. **5**(3), 289–310 (1999). https://doi.org/10.1023/a:1009936607335
36. Nedialkov, N.S., Pryce, J.D.: Solving differential-algebraic equations by Taylor series (i): computing Taylor coefficients. BIT Numer. Math. **45**(3), 561–591 (2005)
37. Obreshkov, N.: Neue quadraturforme. ln: Preussische Akademie der Wissenschaften zu Berlin (1–4), 116–127 (1940)
38. Rackauckas, C., Nie, Q.: DifferentialEquations.jl – a performant and feature-rich ecosystem for solving differential equations in Julia. J. Open Res. Softw. **5**(1), 15–6 (2017). https://doi.org/10.5334/jors.151
39. Rall, L.B.: Automatic differentiation: techniques and applications. Springer (1981)
40. Salvy, B., Zimmermann, P.: Gfun: a Maple package for the manipulation of generating and holonomic functions in one variable. ACM Trans. Math. Softw. (TOMS) **20**(2), 163–177 (1994)
41. Schwarz, D.E., Lamour, R.: Projected explicit and implicit Taylor series methods for DAEs. Numer. Algorithms **88**(2), 615–646 (2021). https://doi.org/10.1007/s11075-020-01051-z

42. Shampine, L., Reichelt, M.: The Matlab ODE suite. SIAM J. Sci. Comput. **18**(1), 1–22 (1997)
43. Shampine, L.F., Corless, R.M.: Initial value problems for ODEs in problem solving environments. J. Comput. Appl. Math. **125**(1), 31–40 (2000)
44. Smoktunowicz, A.: Backward stability of Clenshaw's algorithm. BIT Numer. Math. **42**(3), 600–610 (2002)
45. Söderlind, G., Jay, L., Calvo, M.: Stiffness 1952–2012: sixty years in search of a definition. BIT Numer. Math. **55**(2), 531–558 (2015)
46. Trefethen, L.N.: Approximation theory and approximation practice. SIAM (2019)
47. Zolfaghari, R., Nedialkov, N.S.: An Hermite-Obreschkoff method for stiff high-index DAE. BIT Numer. Math. **63**(1), (2023). https://doi.org/10.1007/s10543-023-00955-1