Blendstrings: an environment for computing with smooth functions

Robert M. Corless
rcorless@uwo.ca
University of Western Ontario
London, Ontario, Canada

ABSTRACT

A “blendstring” is a piecewise polynomial interpolant formed of high-degree two-point Hermite interpolational polynomials on each piece, something like a higher-order cubic spline but more like a chebfun. Blendstrings can be used for accurate representation of smooth functions on a line segment, or more generally a polygonal path, in the complex plane. This paper discusses some properties of these objects including how to evaluate, differentiate, and integrate them efficiently, and describes a prototype Maple implementation. Blendstrings can be differentiated exactly and integrated exactly. Compatible blendstrings can be combined algebraically. Applications to the solution of differential equations and the computation of Mathieu functions and generalized Mathieu eigenfunctions are shown.

CCS CONCEPTS

• **Computing methodologies** → **Representation of mathematical objects**; **Representation of mathematical functions**.

KEYWORDS

Blends, Blendstrings, piecewise polynomial approximation, Taylor series, Hermite interpolational polynomials, Hermite–Obreshkov methods, smooth functions, integration, automatic differentiation

ACM Reference Format:
Robert M. Corless. 2018. Blendstrings: an environment for computing with smooth functions. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym ’XX)*. ACM, New York, NY, USA, 9 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Providing useful tools for dealing with mathematical functions is one of the fundamental tasks of symbolic computation systems. This usually entails various kinds of polynomial or rational approximations, which are used to provide accurate approximations of the values of the function and its derivatives and antiderivatives. The various major systems use different methods, and may even use different methods (hidden from the user) for different functions in the same system. Arbitrary-precision evaluation of the exponential function, for instance, might use argument reduction, rational approximation, and powering. A common model is that the user treats the system’s representation as an oracle, and asks for evaluation of the function at various points. The system may or may not use any previous evaluation points as a shortcut in evaluating at the new point.

In contrast, the Chebfun system [5] uses piecewise Chebyshev polynomials with fast construction methods based on the FFT, and is very efficient and very reliable. The basic model in the Chebfun system, however, is more like simultaneous evaluation at a great many points, although rapid change to Chebyshev series via the FFT is also fundamental. Combination of these functions then becomes a question of numerical combinations of the points. This can be orders of magnitude faster than traditional computer algebra systems can combine or manipulate their typical representations. The ApproxFun system makes even further improvements in speed, accuracy, and generality; see https://juliaapproximation.github.io/ApproxFun.jl/latest/.

This paper describes something similar to Chebfun and ApproxFun, but which uses what I call “blendstrings.” A formal definition will be given in section 2, but, in brief, blendstrings are piecewise polynomials where each polynomial is represented not in the Chebyshev basis but rather as a *blend*. A blend is a two-point Hermite interpolant, typically of quite high degree [11]. To construct a blendstring, one needs Taylor coefficients at several knots, typically lying in a straight line in the complex plane and even more typically on an interval of the real line.

Blendstrings are *not* as good as chebfuns or approxfuns in many ways. They do not have quite as good approximation properties, for instance, which can be seen by their Lebesgue constants, which are $O(\sqrt{n})$ in size whereas those of Chebyshev series are $O(\ln n)$, as the approximation grade $n \to \infty$. Nonetheless blends seem natural in some contexts, such as the numerical solution of Ordinary Differential Equations (ODEs) by high-order marching methods [17]. Such marching methods are of cost polynomial in the number of bits of accuracy requested [14], and are especially interesting for functions whose Taylor coefficients can be computed easily at a given point, i.e. D-finite or holonomic functions [16, 22, 23]. Blendstrings have an arbitrarily high order of continuity, depending only on how many Taylor coefficients are known at each knot. They might be especially useful in arbitrary-precision environments. They have been found to be useful in the study of Mathieu functions [7].

This paper describes a prototype implementation in Maple and some of its capabilities. This implementation is available at https://github.com/rcorless/Blends-in-Maple/blob/main/BlendstringExamples.maple. The implementation uses `evalhf` for speed when possible, but is primarily intended for high precision.

Conference acronym ’XX, June 03–05, 2018, Woodstock, NY
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

2 DEFINITIONS AND BASIC PROPERTIES

Much of this section is based on [11]. Consider an analytic function $f(z)$ with Taylor series coefficients known at $z = a$ and at $z = b$. Convert to the unit interval by introducing a new variable s with $z = a + s(b - a)$.

2.1 The basic formula

The following formula, known already to Hermite [13, p. 4], has the property that the grade $m + n + 1$ polynomial

$$H_{m,n}(s) = \sum_{j=0}^{m} \left[\sum_{k=0}^{m-j} \binom{n+k}{k} s^{k+j} (1-s)^{n+1} \right] p_j + \sum_{j=0}^{n} \left[\sum_{k=0}^{n-j} \binom{m+k}{k} s^{m+1} (1-s)^{k+j} \right] (-1)^j q_j \qquad (1)$$

has a Taylor series matching the given $m+1$ values $p_j = f^{(j)}(0)/j!$ at $s = 0$ and another Taylor series matching the given $n + 1$ values $q_j = f^{(j)}(1)/j!$ at $s = 1$. Putting this in symbolic terms and using a superscript (j) to mean the jth derivative with respect to s gives

$$\frac{H_{m,n}^{(j)}(0)}{j!} = p_j \qquad \text{and} \qquad \frac{H_{m,n}^{(j)}(1)}{j!} = q_j$$

for $0 \le j \le m$ on the left and for $0 \le j \le n$ on the right. This is a kind of interpolation, indeed a special case of what is called *Hermite* interpolation. As with Lagrange interpolation, where for instance two points give a grade one polynomial, that is, a line, here $m + n + 2$ pieces of information gives a grade $m + n + 1$ polynomial. As per [11] this formula can be evaluated in $O(m + n)$ arithmetic operations, and this paper will assume that the Taylor coefficients are known to a fixed working precision, most commonly 15 decimal digits. If one wants to work in higher precision, it will be necessary to start with Taylor coefficients evaluated to that higher precision. The word *grade* means "degree at most". That is, a polynomial of grade (say) 5 is of degree at most 5. But because here the leading coefficient is not visible, we won't generally know the exact degree.

2.2 Definition of a blendstring

A *blendstring* is an ordered finite set of "local Taylor polynomials"

$$\mathrm{LTP}_k(z) := c_{k,0} + c_{k,1}(z - a_k) + c_{k,2}(z - a_k)^2 + \cdots + c_{k,m_k}(z - a_k)^{m_k}$$

for $0 \le k \le M$, together with the line segments $[a_0, a_1]$, $[a_1, a_2]$, ..., $[a_{m_k-1}, a_{m_k}]$, over which one can blend the appropriate Taylor polynomials. The grades $m_k \ge 0$ of the Taylor polynomials are integers. It is not necessary that the leading coefficients c_{k,m_k} be nonzero. It is necessary that the knots a_k be distinct from their predecessor knot and their successor knot (but crossings are permitted otherwise: polygonal paths in the complex plane can return to earlier knots). The local Taylor polynomials may be represented by an array with the knot a_k first and then all the Taylor coefficients: $[a_k, c_{k,0}, \ldots, c_{k,m_k}]$.
Two blendstrings

$$\mathcal{B}_1 = \{[a_k, c_{k,0}, \ldots, c_{k,m_k}]\}_{k=0}^{M}$$

and

$$\mathcal{B}_2 = \{[b_k, d_{k,0}, \ldots, d_{k,n_k}]\}_{k=0}^{N}$$

are *compatible* if $N = M$ and all corresponding knots $a_k = b_k$ and all corresponding grades $m_k = n_k$.

2.3 Approximation theoretic properties

Recall that the Lebesgue function $L(s)$ for a polynomial basis $\phi_j(s)$ for $0 \le j \le N$ is the function $L(s) = \sum_{j=0}^{N} |\phi_j(s)|$. Relative errors δ_j in polynomial coefficients $a_j(1 + \delta_j)$ produce changes $\Delta p(s)$ in the polynomial value. By the triangle inequality, $|\Delta p(s)| \le L(s)\|\mathbf{a}\|_\infty \varepsilon$ if all relative coefficient changes $|\delta_j| \le \varepsilon$. It was claimed in [11] that for a *balanced* blend, that is one where the grade m of Taylor polynomial at the left end is the same as the grade n at the right end, $L(s) \le 2$ on $0 \le s \le 1$ so that the Lebesgue constant, on this interval, is just 2, independently of the degree. On the more usual interval of $[-1, 1]$, the Lebesgue constant $\Lambda_{m,m} \sim 2\sqrt{m/\pi}$ which is not much worse than the optimal growth, which is $O(\ln m)$. The proofs of these claims can be found in [10].
We also claimed in [11] that a double Horner evaluation using IEEE 854 floating-point arithmetic was backward stable on $0 \le s \le 1$ in the following sense: namely, that numerical evaluation with unit roundoff u gave the *exact* answer for a blend with Taylor coefficients p_j and q_j changed to $p_j(1 + \delta_{p,j})$ and $q_j(1 + \delta_{q,j})$ with $|\delta_{\cdot,j}| \le \gamma_{3(m+n)}$ where $\gamma_k = ku/(1 - ku)$. A proof can be found in [10]. This theorem, together with the bound on $L(s)$, guarantees accurate approximation.
Blends and blendstrings do make sense even if the coefficients are exactly known in terms of symbols such as γ, π and the like, but they lose much of their advantage in rapid and stable computation. Assume henceforth that the Taylor series at each knot are approximated to working precision in floating-point complex arithmetic. IEEE-754 double precision is frequently used, but Maple's arbitrary precision arithmetic is sometimes convenient as well. Working at a fixed precision, evaluation of a blend is of cost $O(m + n)$ multiplications at that precision.
The truncation error in approximating $f(z)$ by the blend is given by (in the scaled variable s)

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

Here $\theta \in (0, 1)$ is otherwise unknown. The maximum value of the polynomial $s^{m+1}(1 - s)^{n+1}$ on the interval $(0, 1)$ is attained at the point $s = (m + 1)/(m + n + 1)$ and is $(m+1)^{m+1}(n+1)^{n+1}/(m+n+2)^{m+n+2}$, which if $m = n$ reduces to $2^{-2(m+1)}$. One frequently uses $m = 10$ or more, and so this factor reduces the error by a factor of a million or more, compared to a Taylor series expansion of grade $m + n + 2$ on only one side. In some sense this factor is the real reason blends are interesting.
Approximation by a blendstring is analogous to approximation by cubic splines, though of higher order. If each subinterval of the blendstring is of width h, and each Taylor polynomial is of grade m, then the order of accuracy of the blendstring is $\mathcal{B}(z) - f(z) = O(h^{2m+2})$. More, the accuracy of the derivative is $\mathcal{B}'(z) - f'(z) = O(h^{2m+1})$, and so on, losing one order of accuracy per derivative. The proof follows by standard methods and can be seen using equation (2) once the polynomial $(z - a_n)^{m+1}(z - a_{n+1})^{m+1}$ is expressed using the transformation $z = a_n + sh$. The quite high-order Taylor coefficients that appear will not cause problems for the smooth

functions that are considered here, but can prove troublesome if there are nearby singularities, as expected.

2.4 Evaluation and differentiation of a blend

The paper [11] implemented a double Horner expansion of the formula (1). Several examples were given showing its numerical stability in practice. As stated earlier, the cost of evaluation is linear in the grade of the blend, $O(m+n)$. Since the Horner-like evaluation is simply a pair of for-loops, automatic differentiation is straightforward, and the user can request derivatives of the blend to be delivered with the values. By experiment, rounding errors in the automatic derivatives are similarly small, even though differentiation is infinitely ill-conditioned.

3 THE PROTOTYPE IMPLEMENTATION

The code can be found in the files `Blend.mpl`, `deval.mpl`, and `BlendstringUtilities.mpl`, at https://github.com/rcorless/Blends-in-Maple/blob/main/BlendstringExamples.maple.

3.1 Constructing a blendstring

The basic data structure I chose for the Maple implementation is a two-dimensional **Array**. Specifically, the following command gives an empty blendstring with space for $M+1$ knots and local Taylor polynomials of grade m:

```
B := Array(0..M, 0..m+1 );
```

This limits each m_k to be the same number m, but balanced blends[1] are best anyway for approximation, and this sufficed for my first applications. Provision for variable grades m_k in the same blendstring should be done in a future version. This structure represents the local Taylor polynomials densely in the style described earlier: knot first, coefficients later. The most basic way of constructing a blendstring is simply to create such an object directly.
Constructing a blendstring for a given analytic function $f(z)$ is then a matter of filling the **Array** with the appropriate knots and Taylor coefficients, perhaps generated by the **series** command. For example, the following constructs a blendstring with four knots and local Taylor polynomials of grade 5. This blendstring approximates $\exp(z)$ to better than $5 \cdot 10^{-15}$ over the interval $(-1, 1)$. Figure 1 shows that the error in the 2nd derivative is smaller than 10^{-12}.

```
M := 3;
grade := 5;
Digits := 15;
knots := Array(0..M, k -> ( -1 + 2*k/M) ) ;
f := z->exp(z);
B := Array(0..M, 0..grade+1 );
for k from 0 to M do
  B[k,0] := knots[k];
  S := series( f(z), z=knots[k], grade+1 );
  for j from 0 to grade do
    B[k,j+1] := evalf(coeff(S,z-knots[k],j));
  end do;
end do:
```

[1]A *balanced blend* is one in which $m \approx n$. Exact equality is best, but near equality is almost as good. Highly unbalanced blends are exponentially bad [10].

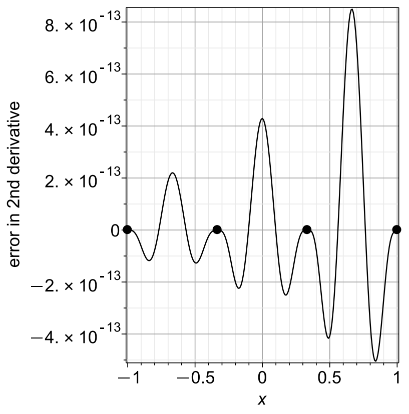

Figure 1: The error in the automatically computed second derivative of the blend for $\exp(z)$ **just constructed with knots at** -1, $-1/3$, $1/3$, **and** 1, **each with Taylor polynomials of grade** $m = 5$.

REMARK 3.1. *The Maple environment variable* **Order**, *or equivalently the third parameter in the call to* **series**, *only sets the "working grade" for* **series** *and does not guarantee that the results are actually that grade of Taylor polynomial. An example is* `series( f(x), x, 6 )` *with* $f(x) = \sin(x)/x$. *The returned answer is* $1 + O(x^4)$. *This difficulty will be ignored in this paper, apart from this mention.*

The next block uses commands that will be described in section 3.4, to plot the error in the 2nd derivative compared to the evaluation of the symbolic derivative.

```
Digits := 30;
y := deval(B, nder = 3, nRefine = 80):
N := upperbound(y)[1]:
diffs := Array(0 .. N, 0 .. 4):
for k from 0 to N do
  diffs[k, 0] := y[k, 0];
  for j to 4 do
    diffs[k,j] := y[k,j]-(D@@(j-1))(f)(diffs[k,0]);
  end do;
end do:
X := Vector(N + 1, j -> y[j-1, 0]):
Y := Vector(N + 1, j -> diffs[j-1, 3]):
eplot := plot(X, Y, colour=black, symbol=point,
                     axes=boxed, gridlines=true);
```

3.2 Arithmetic combinations of compatible blendstrings

Since Taylor polynomials can be added together, it is a simple matter to form linear combinations $\alpha\mathcal{B}_1 + \beta\mathcal{B}_2$. The result is a blend of the same grade as the components (even if some of the leading coefficients of the resulting Taylor polynomials are zero, that is still information about the function that gets used in the blend).

Multiplication requires the Cauchy product, so each Taylor coefficient of the product is

$$p_{k,j} = \sum_{\ell=0}^{j} a_{k,\ell} b_{k,j-\ell} ,$$

for $0 \le j \le m_k$. Unlike in the case of addition, however, this truncated product of two blends is not the blend of the exact product of the two underlying polynomials, because degrees increase with products; nonetheless it is an appropriate approximation to the product of the functions underlying the two blends being multiplied. For the moment, *division* requires that each constant coefficient of the dividend be nonzero, because I do not yet have a method of blending Laurent series. However, the underlying function of the dividend might have a zero on the interval, and this might lead to unexpected results.
These arithmetic combinations have been implemented in this package by the command `zipBlendstrings`, which takes as input an arbitrary binary function $(a, b) \to f(a, b)$ and uses Maple's built-in **`series`** command to carry out the necessary operations. This is efficient (if the efficient routines in the `PolynomialTools` package are used to convert back and forth from Maple polynomials) because **`series`** is in the kernel. Since **`series`** is smart about indeterminate forms, this can be effective.
Simply by using this utility, one can build up a collection of useful blendstrings on a given sequence of knots, starting from the function z which has the value a_k at every knot a_k and derivative 1 and all higher derivatives zero. One could then represent powers of z and polynomials in z as compatible blends by simply applying the operations in sequence. As a simple example, I used the Chebyshev recurrence relation to construct the Chebyshev polynomial $T_6(z)$ as a blendstring on the same knots as the above example. It worked, and was perfectly accurate, which was unsurprising.
A more interesting test occurs when one computes a blendstring on those same knots, $\{-1, -1/3, 1/3, 1\}$, with all Taylor polynomials of grade $m = 5$, for the *rational* function $(1 + z/2)/(1 - z/2)$. This example tries to approximate a rational function on this interval with a piecewise polynomial, where the grade of the polynomial on each of the three subintervals is $m + n + 1 = 11$. The error in this approximation, which is appreciable because rational functions are not well approximated by polynomials, is shown in figure 2.

3.3 Applying a function to a blendstring to get another blendstring

The package contains another function, `mapOntoBlendstring`, which computes $f(\mathcal{B})$ given an operator representation[2] for the function $f(z)$. Again, this uses **`series`** to compute local Taylor polynomials for $f(LTP_k(z))$ about the knots a_k, and gives an approximation to the composition of functions.

3.4 Evaluation and differentiation

Given the capability for efficient and numerically stable evaluation of a blend, all that remains is to decide how to organize the dispatch. I chose not to use something like **`piecewise`**, which is the standard way in Maple to represent a piecewise polynomial; instead I chose

[2]A Maple operator representation for a function $f(z)$ is `z -> f(z)`.

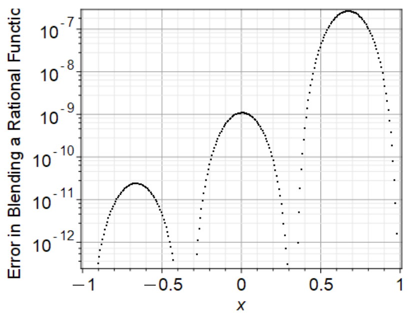

Figure 2: The absolute error $|\mathcal{B}(x_j) - (1 + x_j/2)/(1 - x_j/2)|$, computed at many points. The blendstring $\mathcal{B}$ was constructed by constructing a blendstring for 1 at four equally-spaced knots on the interval $[-1, 1]$, a compatible one for z, a third for $1 + z/2$, a fourth for $1 - z/2$, and finally by dividing the one blendstring by the other.

to model my evaluation of blendstrings on the subroutine `deval` of the Matlab ODE Suite [19]. The main reason for this was speed: by thinking of a blendstring as an "all at once" object, with many function and derivative values already present, one can gain significant speedup. The second reason was that I had erroneously thought that **`piecewise`** was limited to the real line, and the application driving this implementation needs paths in the complex plane. However, that was a failure of imagination on my part, because **`piecewise`** can indeed be used with complex input, so long as the boolean decision functions are correct. But even so, the dispatch model seems geared to "one at a time" inputs, whereas I really did want to "think all at once," which meant to me something like `deval`. The `deval` routine itself in Matlab is reminiscent of `pp` (for "piecewise polynomial") constructs in Matlab, which themselves tend to work "all at once."
The call is either `deval(y,'x'=pt)` to evaluate the blendstring `y` at a single point `pt`, or simply `deval(y)` to evaluate the blendstring at (by default `nRefine=2*`**`Order`**) equally-spaced points on each subinterval. It is frequently the case that one wants to graph an entire blendstring, and using that many interior points is generally sufficient. If not, one can ask for more (or fewer) by the `nRefine=n` option, where n is however many you want.
One reason for evaluating at so many points is for plotting. The current method for plotting is simply to hand the evaluated points to Maple's **`plot`**; this can be inefficient if done with too many conversions back and forth from lists, so it would be well to write a specialized plot routine for the package. This has not yet been done.
To differentiate, one uses the keyword `nder`, as in `deval(y,'nder'=2)` to compute (at all default interior points, as above) $y(x_k)$, $y'(x_k)$, and $y''(x_k)$. The result is a two-dimensional **`Array`**, with knots at all the original knots plus the new evaluations, together with derivatives (not Taylor coefficients) to all requested orders.
An important caveat. Blends provide good approximations only on the segment $[a_k, a_{k+1}]$ which gets mapped to $0 \le s \le 1$, or very near in the complex plane to that segment. Away from that segment, the truncation error grows very rapidly, like $|s|^{m+n+1}$. So if one is using a blendstring to approximate a function in the complex plane, but not actually *on* the segment, getting accurate answers is

not easy and may not be possible. Further, if the segments of the blendstring follow a polygonal path or even make loops, it may not be evident which piece of the blendstring would be best to use to evaluate the function with. In practice I have only used the `nRefine` option in cases such as this, which guarantees that each evaluation point is precisely located exactly on one of the defining segments.

3.5 Integration

It is integration for which blends and blendstrings truly shine. There is a formula reported in [11] for integrating a blend.

$$\int_{s=0}^{1} H_{m,n}(s)\,ds = \frac{(m+1)!}{(m+n+2)!}\sum_{j=0}^{m}\frac{(n+m-j+1)!}{(j+1)\,(m-j)!}\,p_j + \frac{(n+1)!}{(m+n+2)!}\sum_{j=0}^{n}\frac{(n+m-j+1)!}{(j+1)\,(n-j)!}\,(-1)^j\,q_j\,. \tag{3}$$

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 this becomes a kind of numerical quadrature. From this formula, one can construct an "exact" blend for the indefinite integral across the blend because now the Taylor coefficients *for the integral* are known at each end. It is a simple matter to propagate constants along from one end of a blendstring to the other to construct an "exact" indefinite integral of the original blendstring.
This is an exact integral *for the blendstring*. This gives an *approximation* to the integral of the underlying function $f(z)$ that the blendstring approximates. Theory predicts that the results will be most satisfactory for balanced blends [10].

Accuracy and stability. Because the formula is principally of use with floating-point approximations to the Taylor coefficients, the following theorem is useful. Let the bold symbol $\mathbf{1}$ represent the series with all coefficients equal to 1, and the bold symbol $(-\mathbf{1})^{\mathbf{k}}$ represent the series with all coefficients alternating in sign starting with $(-1)^0 = 1$.

Theorem 3.1. *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 $\int_0^1 H_{m,n}(\mathbf{1}, (-\mathbf{1})^{\mathbf{k}}) \max|\Delta p_j|, |\Delta q_j|$.*

The proof is immediate by using the linearity of the blend and the linearity of the integral. Further, the integral can be explicitly computed:

$$\int_0^1 H_{m,n}(\mathbf{1}, (-\mathbf{1})^{\mathbf{k}}) = 2\Psi(n+m+3) - \Psi(m+3) - \Psi(n+3) + \frac{n+m+4}{(n+2)(m+2)}\,. \tag{4}$$

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 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.

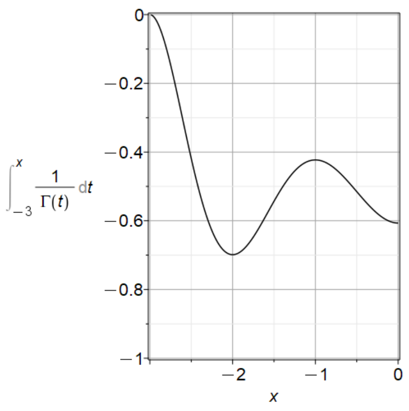

Figure 3: A plot of a blendstring for the indefinite integral of $1/\Gamma(x)$.

An example. Here is an example, a quadrature for the function $1/\Gamma(s)$ on the interval $-3 \le s \le 0$. The code below uses a blend of the known series at the negative integers, and grade 7 Taylor polynomials at each knot:

```
Digits := 15;
knots := [seq(-3 + k, k = 0 .. 3)]:
grade := 7:
B := Array(0 .. numelems(knots) - 1, 0 .. grade+1):
for k from 0 to numelems(knots) - 1 do
  B[k, 0] := knots[k + 1];
  S := series(1/GAMMA(s),s=knots[k + 1],grade+1);
  for j to grade+1 do
    B[k,j] := evalf(coeftayl(S,s=knots[k+1],j-1));
  end do;
end do:
Y := CodeTools:-Usage( intBlend(B) ):
CodeTools:-Usage(evalf(Int(1/GAMMA(s),s=-3..0)));
evalf(Y[numelems(knots) - 1, 1]);
```

Maple's built-in numerical integrator [12] gives −0.606607588776539 while the blend integration gives −0.606607588783124. Using grade 8 Taylor polynomials instead gives better agreement, and using grade 10 agrees to 17 decimals if we work in 30 Digits. It is not fair to compare the timing of these (`intBlend` is much faster on this example), because the series coefficients of $1/\Gamma(s)$ are known symbolically at negative integers and essentially free to evaluate numerically, while the general-purpose built-in numerical integrator does not take advantage of that information for this specific problem. On other examples the built-in integrator can be faster. On the other hand, the integral blend above can be evaluated efficiently at many points between −3 and 0; it has effectively computed the *indefinite* integral. This could be advantageous for some situations.

```
y := CodeTools:-Usage(deval(Y, nRefine = 80)):
npts := upperbound(y)[1]:
```

The evaluation of the blendstring for the antiderivative of $1/\Gamma(x)$ took 31ms and thereafter the plot, shown in Figure 3, took no measureable time. In contrast, the plot of the built-in version took nearly two seconds on the same machine.

4 APPLICATIONS

There are many methods for approximating functions, and many applications for approximation of functions [21]. The real question is, are there any applications for which blendstrings might be especially suited? Their main benefits are a high degree of smoothness: if all the grades of Taylor coefficients are m, then the resulting blendstring is in C^m. They require, however, derivatives (Taylor coefficients) at all knots. This suggests that they will be useful for marching methods used to solve IVP for ODE *at very high precision*. They are indeed used in this way for DAE [17]. In this section we will pursue this application. I will end by discussing the use of the integration method for computing generalized Mathieu functions, which are needed at a double eigenvalue of the Mathieu equation.

4.1 Collocation methods for ODE

In this section I describe the method I implemented for [7] and [8], which uses the blendstring environment described in this present paper.

One of the best codes for the numerical solution of boundary value problems (BVP) for ODE is the FORTRAN code COLSYS [1], with theoretical foundations described by the analysis in the SIAM Classic book [2]. The method that code uses is *collocation*, which requires that the differential equation be satisfied exactly at certain points, called collocation points, in each subinterval.

Their original code [1] used B-splines as a piecewise polynomial interpolant, but later the authors examined Hermite and monomial bases [3] and found them to be superior. Those bases were not the same as used here, because they incorporated Runge–Kutta method information as well. There are significant simplifications that occur when, instead of Runge–Kutta information, one uses actual Taylor coefficients.

The automatic generation of Taylor series coefficients for linear differential equations with polynomial coefficients is well understood. For instance, in Maple, one can use the `diffeqtorec` routine from the `gfun` package [18] to generate the recurrence relations. There are other methods, some even implemented in FORTRAN [9]. Generation of Taylor coefficients costs at most $O(m^2)$ floating-point operations in general, but for D-finite or holonomic systems the cost is only $O(m)$ [16, 22, 23].

It is efficient to use Taylor coefficients generated at *both* ends of the marching step, and blend these coefficients together. This kind of method is called an Hermite–Obreschkov method, after the French 19th century mathematician Charles Hermite and the Bulgarian 20th century mathematician Nikola Obreschkov. Methods like this have been used with great success for differential-algebraic equations (DAE) [17] and recent work has uncovered a class of conjugate symplectic such methods [15]. The main advantage to this method over an explicit Taylor series method is that by combining the terms at either end one has a method of order $2m$ instead of just m, at very little extra cost.

For simplicity of presentation, assume that we are solving a linear second-order scalar ODE, say $y'' + a(x)y' + b(x)y = g(x)$. Further assume that Taylor series for the functions $a(x)$, $b(x)$, and $g(x)$ are available at working precision on demand. What follows is one convenient and accurate method for marching from one knot to the next, by using collocation.

The method assumes that Taylor series coefficients have been generated at the current knot, say a_n, and are considered to be"known". Specifically, suppose to start with that a Taylor polynomial of grade m for the desired solution is known at this knot.

Suppose also that we have chosen a tentative next knot, $a_{n+1} = a_n + h$. If the variable z were time, this would be a time step. The stepsize h is tentative at this point. Now generate Taylor coefficients up to grade m for two independent solutions, satisfying (for one solution)

$$y(a_{n+1}) = 1 \text{ and } y'(a_{n+1}) = 0 \tag{5}$$

and (for the complementary solution)

$$y(a_{n+1}) = 0 \text{ and } y'(a_{n+1}) = 1\ . \tag{6}$$

Next, *blend* the known coefficients at a_n with these independent solutions in the following way. Form a blend of the known coefficients at a_n with the *zero* Taylor series at a_{n+1}. Call the result $L(z)$. Form a blend of the first series above at a_{n+1} with the *zero* Taylor series at a_n. Call the result $C(z)$. Form a blend of the second series above with the *zero* Taylor series at a_n and call the result $S(z)$. The desired solution will then be a linear combination of these three: say $y = A\,C(z) + B\,S(z) + L(z)$. This uses the linearity of the equation, and the linear dependence of blends on their constituent Taylor coefficients.

Now use *collocation* at the two[3] points $a_n + h/4$ and $a_n + 3h/4$ (these are Chebyshev–Lobatto points, which have good properties; general such points are available for work with higher-order equations) to give us two equations in the two unknowns A and B. That is, compute the residuals

$$\begin{aligned} r_L(z) &:= \quad L'' + a(z)L' + b(z)L - g(z) \\ r_C(z) &:= \quad C'' + a(z)C' + b(z)C - g(z) \\ r_S(z) &:= \quad S'' + a(z)S' + b(z)S - g(z) \end{aligned} \tag{7}$$

at those two collocation points, and set the residual for y to zero at those two points:

$$\begin{aligned} 0 &= A\, r_C(a_n + h/4) + B\, r_S(a_n + h/4) + r_L(a_n + h/4) \\ 0 &= A\, r_C(a_n + 3h/4) + B\, r_S(a_n + 3h/4) + r_L(a_n + 3h/4)\ . \end{aligned} \tag{8}$$

One may solve this two-by-two linear system by any method in order to find the coefficients A and B. This system is nonsingular because the solutions are linearly independent at the right endpoint. Experience shows that the equations are well-scaled and well-conditioned, but this needs a proper analysis as in [4]. One might expect that meshes with widely varying mesh sizes, or especially dense meshes, might cause problems. However, today we have a resource that was inconvenient in the old FORTRAN codes: namely, we may use higher precision if necessary.

Now *sample* the residual (after all, we have a blend for the putative solution, and so we can evaluate it and its derivatives wherever we choose) at $a_n + h/2$, which is (asymptotically as $h \longrightarrow 0$) the location of the maximum. Asymptotically as the stepsize $h \longrightarrow 0$ the residual has the form

$$\begin{aligned} r(z) &= y'' + a(z)y' + b(z)y - g(z) \\ &= h^{2m} K s^{m-1}(s - 1/4)(s - 3/4)(s - 1)^{m-1} + \text{h.o.t.}\ , \end{aligned} \tag{9}$$

[3] Two, because this example equation is second order.

where $s = (z - a_n)/h$, “h.o.t" means “higher-order terms", and K is a high-order Taylor coefficient evaluated at a point between a_n and a_{n+1}. This truncation error is proportional to h^{2m} and the maximum of the polynomial $s^{m-1}(s-1/4)(s-3/4)(s-1)^{m-1}$ occurs at $s = 1/2$. If the sampled residual is smaller than the user’s tolerance, accept the step and move on; if not, reject the step and adjust the prediction for the tentative new a_{n+1} and try again. Standard heuristics can be used here.

This presents the essence of collocation. One can see why this might be attractive in a high-precision environment. Instead of having an mth order method with Taylor polynomials of grade m, one has a $2m$th order method and moreover the use of a blend is essential to get this. It is really the smallness of the error coefficients, which is determined by the infinity norm of the s-polynomial, that makes this attractive.

This is an *implicit* method, which is appropriate if the ODE is *stiff* [20], and can help for oscillatory problems as well.

4.2 Stability of the method for oscillatory ODE

When one tries this method *symbolically* on the simple harmonic oscillator $\ddot{y} + \omega^2 y = 0$, it is possible to discover an interesting stability limitation on the allowable stepsize.

To begin, exactly solve, by hand or otherwise, the simple harmonic oscillator with initial conditions $y(0) = y_0$ and $y'(0) = y_1$. The answer can be expressed as $y(t) = y_0 \cos \omega t + y_1 \sin(\omega t)/\omega$. 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

$$\begin{bmatrix} Y_0 \\ Y_1 \end{bmatrix} = \begin{bmatrix} \cos \omega h & \frac{\sin \omega h}{\omega} \\ -\omega \sin \omega h & \cos \omega h \end{bmatrix} \begin{bmatrix} y_0 \\ y_1 \end{bmatrix} . \tag{10}$$

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 $S_m(\nu)$ (easily computed for any fixed m) of $\nu = \omega h$ in place of $\cos \nu$ and $\sin \nu$:

$$\mathbf{A}_m := \begin{bmatrix} C_m(\nu) & \frac{S_m(\nu)}{\omega} \\ -\omega S_m(\nu) & C_m(\nu) \end{bmatrix} . \tag{11}$$

Because it turns out that $C_m^2 + S_m^2 = 1$, the matrix has characteristic polynomial $\lambda^2 - 2C_m(\nu)\lambda + 1$, implying that the product of its two eigenvalues is 1. The eigenvalues are $C_m(\nu) \pm iS_m(\mu)$. However, the eigenvalues *both* have magnitude 1 if and only if $|C_m(\nu)| \le 1$. 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 one of them, say λ_1, will be larger than 1 in modulus and therefore the lengths of the vectors $[y_k, y'_k]$ will 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 $\nu < \nu^*$, or (approximately) $h < \pi/\omega$. For high frequencies ω one would thus seem to have to take very small timesteps.

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	$-\frac{2(33\nu^6-4059\nu^4+84480\nu^2-184320)}{3(3\nu^6+146\nu^4+5120\nu^2+122880)}$	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

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 $\nu^*/\pi \approx 1 - 10^{-7}$.

So it would seem that there is a stability restriction akin to the stepsize restrictions for explicit methods for stiff problems [20]. 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 $1 + O(10^{-6})$. 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 million steps were taken! For higher m, this 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 ν, no matter how large one takes m. This is because the method is not A-stable [20], 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 ten thousand steps of the method one should be able to take any $h < 10\pi/\omega$; this *is* a stability restriction, but is ten times better than the technically correct one of $h < (1 - 4.29 \cdot 10^{-45})\pi/\omega$. And, in any case, one will want stepsizes small enough to resolve the actual oscillations.

4.3 Computing generalized Mathieu functions

We actually used this method, in practice [7]. We have just submitted a paper on a problem in hemodynamics where the code described in that paper is providing us with the solutions for blood flow in a tube of elliptic cross-section, using Mathieu functions. The *reason* we did this is that no existing implementations of Mathieu functions, to our knowledge, could handle the case of double eigenvalues. For instance, we needed to compute the Mathieu functions $\mathrm{ce}_0(z; q)$ and $\mathrm{ce}_2(z; q)$ together with the corresponding modified Mathieu functions for various purely imaginary values of the parameter q. But for $q = 1.468\ldots i$, the two eigenfunctions $\mathrm{ce}_0(z, q)$ and $\mathrm{ce}_2(z, q)$ *coalesce*. We therefore needed also to find the *generalized* eigenfunction $u(z)$ and its corresponding “modified" generalized eigenfunction $U(z)$ in order to express the solution to our problem. This requires solving

$$u'' + (a - 2q\cos 2z)u + \mathrm{ce}_0(z;q) = 0\,. \tag{12}$$

A simple way to do this, if $\mathrm{ce}_0(z;q)$ is expressed as a blendstring, is to compute (on a compatible blendstring) both linearly independent solutions of the Mathieu equation: in the terminology of the DLMF (Chapter 28), these are $w_I(z,a,q)$ (which is really $\mathrm{ce}_0(z;q)$) and $w_{II}(z,a,q)$. We then form the Green's function

$$G(z,\zeta) = w_I(\zeta)w_{II}(z) - w_{II}(\zeta)w_I(z)$$

and integrate (as blendstrings) to get

$$u(z) = -w_{II}(z)\int_{\zeta=0}^{z} w_I(\zeta)\mathrm{ce}_0(\zeta)\,d\zeta + w_I(z)\int_{\zeta=0}^{z} w_{II}(\zeta)\mathrm{ce}_0(\zeta)\,d\zeta\,.$$

This works very well. See [7]. For the hemodynamics work using this method, which heavily uses the convenience of both the integration and differentiation of the solutions, see [8].

For most ranges of the parameters, double precision suffices. For instance when $q = 1.468\ldots i$, the semiminor axis $\beta = 0.151$, the semimajor axis $\alpha = 0.168$, and with grade $m = 15$ Taylor polynomials (so an order 30 method), the solver takes 395ms to take 7 steps across the interval[4] $[0, 2\pi]$ to compute $\mathrm{ce}_0(\eta;q)$. It takes 130ms to take 3 steps across the (vertical) interval $[0, \xi_0\, i]$ where $\xi_0 = 1.485$ is the parameter value at the edge of the ellipse. Because of the doubly-exponential growth of $\mathrm{Ce}_0(\xi,q) = \mathrm{ce}_0(i\xi,q)$, $|\mathrm{Ce}_0(\xi_0,q)| = 4.7\cdot 10^8$. Using the blendstrings and `intBlend` to construct the generalized $u(\eta)$ shown in figure 4 takes no measureable further time. All computations done in Maple 2022.1 running on a Microsoft Surface Pro 7 with Intel® Core™ i7 1065G7 1.30GHz with 4 cores and 8 logical processors.

If, however, the eccentricity ε approaches 0, the problem requires very high precision. This seems paradoxical because circles ought to be easier than ellipses. But the coordinate transformation used, with confocal elliptical coordinates, becomes singular here and we have $\xi \to \infty$. Given the doubly exponential growth of Modified Mathieu functions, this becomes problematic very quickly. Already by $\varepsilon = 0.02$ one requires hundreds of digits of precision. We successfully used 100 Digits and $m = 80$ (giving a method of order 160) in that case; solution still only took 4 steps for ce_0, 8 steps for Ce_0, and less than ten seconds of real time.

5 CONCLUDING REMARKS

One feature needed for the handling of derivative discontinuities is the ability to have knots with different grades of Taylor polynomials; in particular it should be possible to have just a constant (grade zero) function value at a place where the derivative does not exist. In Figure 5 we see what happens if we try to construct a blendstring with grade 20 Taylor polynomials on the four equally spaced knots -1, $-1/3$, $1/3$ and 1 on $[-1, 1]$ for the absolute value function $f(x) = |x|$. Polynomials cannot turn sharp corners. However, the blendstring does surprisingly well, in that it maintains convexity for this function[5]. It would be best, however, to be able to insert a knot of grade zero exactly at the location of the derivative discontinuity. Unbalanced blends, however, have exponentially worse numerical properties, so some kind of "multi-blend" structure with differing series possible to the left or right of the knot might be better.

It should be possible to construct an even more accurate (and therefore efficient) kind of blend, using two-point Padé approximation. The general global Hermite–Padé construction is well-known [6], but if the method were to be used to construct *piecewise* rational approximations with a high degree of continuity at the knots, the result may be quite interesting. Preliminary experiments are encouraging. Another generalization, to Puiseux and to Laurent series approximations on each piece, also seems worth pursuing. Use of blendstrings for matrix functions might also be interesting.

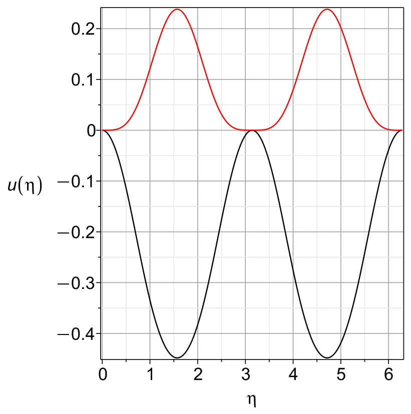

Figure 4: The generalized eigenfunction obtained when $\mathrm{ce}_0(x;q)$ and $\mathrm{ce}_2(x;q)$ coalesce at $q = 1.468\ldots i$. Real part in black, imaginary part in red. The solution was computed by integrating the blendstrings needed in the Green's function for the problem.

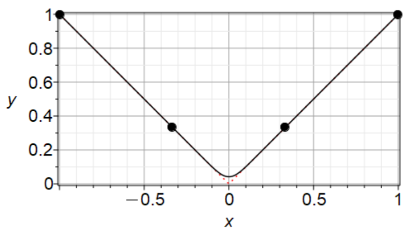

Figure 5: A blendstring on four equally-spaced knots with all Taylor polynomials of grade $m = 20$, attempting to approximate $y = |x|$, which is drawn with red dashed line. Polynomials cannot handle corners well, although at least this approximation maintains convexity.

[4]Integration could have been done on $[0, \pi]$ and saved half the time, but some of the subsequent computations concern integrals around a full period, and computation across the interval $[0, 2\pi]$ made some of the bookkeeping for that simpler.

[5]It works well all the way up to grade 510 Taylor polynomials. The first visible failure (overflow, causing gaps in the graph near the knots) happens when the grade is 511. At that point the binomials in Hermite's formula are of size $1.12\cdot 10^{306}$. Even so, the corner is visibly sharp, and perfectly placed.

ACKNOWLEDGMENTS

This is an outgrowth of work done jointly with Erik Postma, and with Mair Zamir & Chris Brimacombe. I thank Erik Postma and Michael Monagan for comments on earlier drafts. I also thank John C. Butcher for teaching me about the contour integral method (likely originally due to Hermite) for constructing interpolational polynomials. This work was supported by NSERC under RGPIN-2020-06438 and by the grant PID2020-113192GB-I00 (Mathematical Visualization: Foundations, Algorithms and Applications) from the Spanish MICINN.

REFERENCES

[1] Uri Ascher, J Christiansen, and Robert D Russell. 1979. COLSYS–A collocation code for boundary-value problems. In *Codes for Boundary-Value problems in ordinary differential equations*. Springer, 164–185.

[2] Uri Ascher, Robert Mattheij, and Robert D Russell. 1995. *Numerical solution of boundary value problems for ordinary differential equations*. SIAM.

[3] Uri Ascher, Steven Pruess, and Robert D Russell. 1983. On spline basis selection for solving differential equations. *SIAM journal on Numerical Analysis* 20, 1 (1983), 121–142.

[4] Georg Bader and Uri Ascher. 1987. A new basis implementation for a mixed order boundary value ODE solver. *SIAM journal on scientific and statistical computing* 8, 4 (1987), 483–500.

[5] Zachary Battles and Lloyd N. Trefethen. 2004. An Extension of MATLAB to Continuous Functions and Operators. *SIAM Journal on Scientific Computing* 25, 5 (Jan. 2004), 1743–1770. https://doi.org/10.1137/s1064827503430126

[6] Bernhard Beckermann and George Labahn. 1992. A uniform approach for Hermite Padé and simultaneous Padé approximants and their matrix-type generalizations. *Numerical Algorithms* 3, 1 (1992), 45–54.

[7] Chris Brimacombe, Robert M. Corless, and Mair Zamir. 2021. Computation and applications of Mathieu functions: A historical perspective. *SIAM Rev.* 63, 4 (Jan. 2021), 653–720. https://doi.org/10.1137/20m135786x

[8] Chris Brimacombe, Robert M. Corless, and Mair Zamir. 2023. Elliptic cross sections in blood flow regulation. *in preparation* (2023).

[9] YF Chang and George Corliss. 1994. ATOMFT: solving ODEs and DAEs using Taylor series. *Computers & Mathematics with Applications* 28, 10-12 (1994), 209–233.

[10] Robert M. Corless. 2023. Blends have decent numerical properties. *Maple Transactions* 3, 1 (Feb. 2023). https://doi.org/10.5206/mt.v3i1.15890

[11] Robert M. Corless and Erik J. Postma. 2021. Blends in Maple. In *Communications in Computer and Information Science*. Springer International Publishing, 167–184. https://doi.org/10.1007/978-3-030-81698-8_12

[12] Keith O Geddes and Gregory J Fee. 1992. Hybrid symbolic-numeric integration in MAPLE. In *Papers from the international symposium on Symbolic and algebraic computation*. 36–41.

[13] Charles Hermite. 1873. *Cours d'analyse de l'École polytechnique*. Vol. 25. Gauthier-Villars.

[14] Silvana Ilie, Gustaf Söderlind, and Robert M Corless. 2008. Adaptivity and computational complexity in the numerical solution of ODEs. *Journal of Complexity* 24, 3 (2008), 341–361.

[15] Francesca Mazzia and Alessandra Sestini. 2018. On a Class of Conjugate Symplectic Hermite–Obreshkov One-Step Methods with Continuous Spline Extension. *Axioms* 7, 3 (Aug. 2018), 58. https://doi.org/10.3390/axioms7030058

[16] Marc Mezzarobba. 2012. A note on the space complexity of fast D-finite function evaluation. In *Int. Workshop on Computer Algebra in Scientific Computing*. Springer, 212–223.

[17] Nedialko S Nedialkov and John D Pryce. 2005. Solving differential-algebraic equations by Taylor series (I): Computing Taylor coefficients. *BIT Numerical Mathematics* 45, 3 (2005), 561–591.

[18] Bruno Salvy and Paul Zimmermann. 1994. Gfun: a Maple package for the manipulation of generating and holonomic functions in one variable. *ACM Transactions on Mathematical Software (TOMS)* 20, 2 (1994), 163–177.

[19] Lawrence F Shampine and Mark W Reichelt. 1997. The MATLAB ODE suite. *SIAM journal on scientific computing* 18, 1 (1997), 1–22.

[20] Gustaf Söderlind, Laurent Jay, and Manuel Calvo. 2015. Stiffness 1952–2012: Sixty years in search of a definition. *BIT Numerical Mathematics* 55, 2 (2015), 531–558.

[21] Lloyd N Trefethen. 2019. *Approximation Theory and Approximation Practice*. SIAM.

[22] Joris van der Hoeven. 1999. Fast evaluation of holonomic functions. *Theoretical Computer Science* 210, 1 (Jan. 1999), 199–215. https://doi.org/10.1016/s0304-3975(98)00102-9

[23] Joris van der Hoeven. 2001. Fast evaluation of holonomic functions near and in regular singularities. *Journal of Symbolic Computation* 31, 6 (2001), 717–744.

Received 31 January 2023