Blendstrings: An environment for computing with smooth functions

Robert M. Corless

17 August 2023

Maple Transactions

Maple Transactions

an open access journal with no page charges

mapletransactions.org

Remembering Peter Baddoo

Log lightning computation of capacity and Green's functions [Link] by Trefethen & Baddoo

Peter's video abstract for that Maple Transactions paper has set the bar very high for all subsequent video abstracts. See also the "Interview with the authors" in the same issue.

What this talk is about

These slides (which include links) are available at
`https://rcorless.github.io/`

1. What's a "blend?" (a two-point Hermite interpolational polynomial, typically of high order)
2. What's a "blendstring?" (a smooth **piecewise** polynomial where the pieces are blends that share derivative information)
3. Blends can be evaluated in linear time
4. Blends are componentwise backward stable
5. Blends have decent Lebesgue constants
6. Blendstrings can be integrated and differentiated accurately
7. There are companion matrix pencils for blends (which I outline on the YouTube video version of this talk, but I won't talk about them today)

Some of my papers related to this topic

0. Piers Lawrence & RMC, "Numerical stability of barycentric Hermite rootfinding" [Link to ACM Digital Library version] SNC 2011
1. RMC & Erik Postma, "Blends in Maple" [arXiv Link] Maple Conference 2020
2. Chris Brimacombe, RMC, & Mair Zamir, "Computation and applications of Mathieu functions: A historical perspective" SIAM Review 2021 [Open Access Link, courtesy MZ]
3. RMC, "Sobre la iteración cúbica inversa", Gaceta de la Real Sociedad Matematica Española, 2021 [arXiV link] English version
4. RMC, "Blends have decent numerical properties" [Link] Maple Transactions 2023
5. Chris Brimacombe, RMC, & Mair Zamir, "Elliptic cross sections in blood flow regulation" Mathematics, Science, & Industry July 2023 [Open Access Link, courtesy MZ]

I wrote new code in Maple for this

I wrote an Hermite–Obreshkov ODE solver in Maple in order to approximate both Mathieu and modified Mathieu functions and provide a spectrally convergent expansion for the flow of blood in an elliptic cross-section blood vessel:

$$v(\xi,\eta) = \sum_{m\geq 0} b_{2m}\mathrm{ce}_{2m}(\eta)\mathrm{Ce}_{2m}(\xi) \tag{1}$$

and thus solve the original model equation. The code I wrote uses *blendstrings* to approximate the Mathieu functions $\mathrm{ce}_{2m}(\eta)$ and the modified Mathieu functions $\mathrm{Ce}_{2m}(\xi)$ to arbitrary precision.

But why did I write my own special-purpose IVP solver?

- The Mathieu equations have *double eigenvalues*. All the codes that I knew of could not handle the double eigenvalue problem explicitly.
- Mathieu functions are useful: spectral convergence so only six terms needed for double precision accuracy
- I wanted an independent method whose solutions could be verified *a posteriori*
- Purely imaginary $q = i\rho\omega d^2/4$ is needed [2.]
- The case of "near circularity" is actually *very hard* and one needs high precision.

Could there be other uses for blendstrings?

- Efficient high-accuracy solution of IVP (or BVP) for D-finite ODE
- Solving *delay* differential equations by the method of steps[1].
- Ned Nedialkov and John Pryce have already implemented a similar method in a quite general way, for solving DAE. Their code `DAETS` works well. Perhaps some experimental features of this Maple code could influence future development of `DAETS`.
  `http://www.cas.mcmaster.ca/~nedialk/daets/`

[1] I haven't tried this yet but I will soon. I need to implement discontinuity handling first.

Blends and Blendstrings

To implement this code, I first wrote (with Erik Postma) an **efficient** and **numerically stable** evaluator for what we call "blends" (arbitrary degree two-point Hermite interpolational polynomials), together with routines for manipulating them: integration, differentiation, rootfinding, addition, multiplication, etc. [1]

A "string of blends" is a particular kind of piecewise interpolational polynomial, that has some interesting properties, especially in the context of ODE solving.

Blendstrings are analogous to cubic splines[2], but much smoother. Approximations on each subinterval are grade[3] $m + n + 1$ (usually $m = n$) and are m-times differentiable at the knots.

---

[2] Splines are not blends because the derivatives at the knots are approximations, but pure Hermite cubic interpolation with exact derivatives $f'(x_i)$ at the knots is a blend.

[3] degree at most

Blends

Suppose that we know some Taylor coefficients of a function at two distinct points, say $z = a$ and $z = b$. Then put $z = a + s(b - a)$ and the interval $0 \leq s \leq 1$ determines a line segment in the z-plane.

Then (Hermite, Cours d'Analyse 1873)

$$H(s) = \sum_{j=0}^{m} p_j \sum_{k=0}^{m-j} \binom{n+k}{k} s^{k+j} (1-s)^{n+1} + \sum_{j=0}^{n} (-1)^j q_j \sum_{k=0}^{n-j} \binom{m+k}{k} s^{m+1} (1-s)^{k+j} \qquad (2)$$

has $H^{(j)}(0)/j! = p_j$ for $0 \leq j \leq m$ and $H^{(j)}(1)/j! = q_j$ for $0 \leq j \leq n$. Here differentiation is wrt s so one has to be careful about bookkeeping.

This is a "two-point Hermite interpolational polynomial," or "blend" for short, because it blends the Taylor series at either end together to give an approximant on the interval between.

Double Horner

When evaluating a polynomial $p(x) = c_0 + c_1 x + c_2 x^2 + \cdots + c_m x^m$ one usually writes it in Horner form:
$p(x) = c_0 + x(c_1 + x(c_2 + \cdots + x c_m) \cdots)$. A similar thing can be done here for the *double* sum. The key step is the recurrence for the half-sums which involve terms a_k analogous to powers x^k: $a_0 = 1$ and

$$a_k \leftarrow (n + k) \cdot \sigma \cdot a_{k-1}/k \tag{3}$$

which not only computes powers of σ (which is either s or $1 - s$ depending on which double sum we're doing) but breaks the binomial coefficients down using their recurrence relation. See [1].

We also need partial sums of these, and to incorporate the coefficients,

Numerical stability

Blends are *ridiculously* good numerically. The presence of those potentially large binomial coefficients suggests the opposite, but the doubly-recursive Horner implementation has been used successfully for degrees up to about 1000, even for very hard-to-approximate functions. That is, the ability of a polynomial to approximate whatever it is usually fails first, before the blend has any numerical instability.

There's a couple of reasons for that success.

Backward stability result

- double Horner gives the *exact* value of a blend with Taylor coefficients $p_j(1+\theta_j)$ and $q_j(1+\psi_j)$, where each $|\theta_j| \le \gamma_{3M+N}$ and $|\psi_j| \le \gamma_{3M+N}$, with $M = \max(m,n)$ and $N = \min(m,n)$.
- proof uses the nonnegativity of $s^a(1-s)^b$ on $0 \le s \le 1$

Zero coefficients $p_j = 0$ and $q_j = 0$ are not disturbed. This is a *very strong* backward stability result (cf. those of Alicja Smoktunowicz for the Clenshaw–Curtis algorithm).

Here $\gamma_n = nu/(1-nu) = nu + O(u^2)$ where u is the unit roundoff. For double precision, $u \approx 10^{-16}$. See e.g. Higham's *Accuracy and Stability of Numerical Algorithms*. This proof and all its details are in the Maple Transactions paper [4].

A grade 1000 example

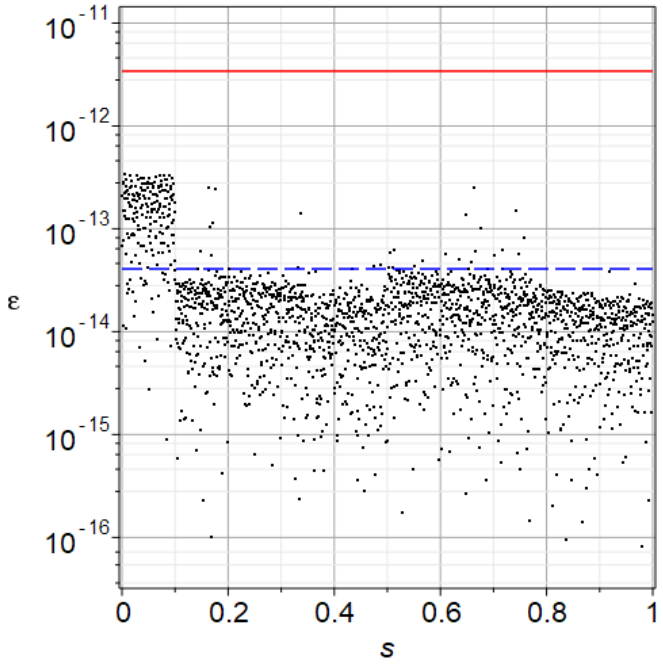

Figure 1: The backward error for a random example, computed in double precision. The Taylor coefficients p_j and q_j were pseudorandomly generated; $m = 368$ and $n = 999 - m = 631$ so the grade is $m + n + 1 = 1000$. The red line is the theoretical bound γ_{3M+N}, the blue dashed line is a crude random walk model of the error $\gamma_{\sqrt{3M+N}}$, and the black dots are the actual backward error computed by the Oettli–Prager theorem.

Lebesgue constants on $-1 \le x \le 1$

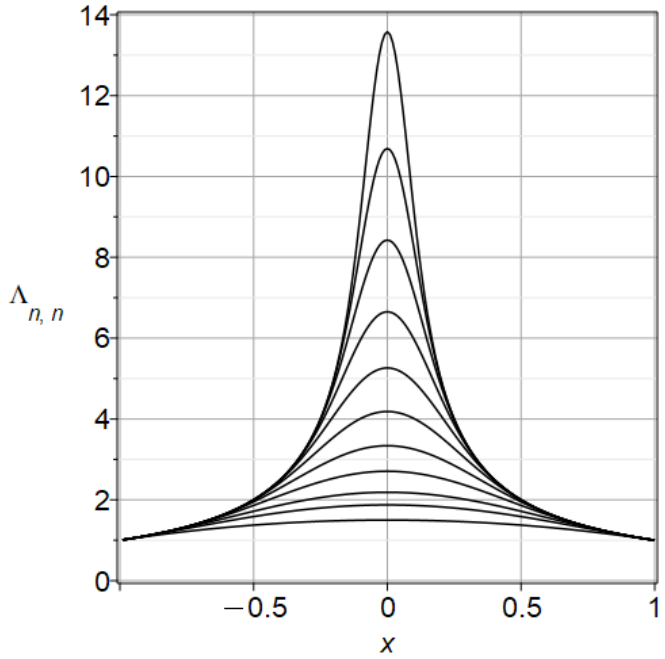

Figure 2: The Lebesgue function of balanced blends on $[-1,1]$ grows like $2\sqrt{m/\pi}$ where the grade is $2m+1$. We plot the function for $m = 1$, 2, 3, 5, 8, 13, 21, 34, 55, 89, and 144. Lebesgue constants *must* be unbounded on $[-1,1]$; optimal is $2\ln(2m+1)/\pi + O(1)$, like Lagrange interpolation on Chebyshev nodes. But $\sqrt{m}$ is pretty decent. For $m = 144$ this is 13.5 vs 3.6. See also `https://www.chebfun.org/examples/approx/LebesgueConst.html`.

On $0 \leq s \leq 1$ the Lebesgue constant is 2

The Lebesgue *function*—which the Lebesgue constant is a bound for—gives a bound for the condition number. For blends the Lebesgue function is exactly the polynomial you get with all series coefficients 1 at the left and all coefficients $(-1)^j$ on the right:

$$L_{m,n}(s) = \sum_{j=0}^{m} \sum_{k=0}^{m-j} \binom{n+k}{k} s^{k+j} (1-s)^{n+1} + \sum_{j=0}^{n} \sum_{k=0}^{n-j} \binom{m+k}{k} s^{m+1} (1-s)^{k+j} \tag{4}$$

We can show that on $0 \leq s \leq 1$, $L_{m,m}(s) < 2$ and indeed $L_{m,m}(s) \leq L_{m,m}(1/2) = 2 - 2^{-2m+1}\binom{2m+2}{m+1} \sim 2 - 2/\sqrt{\pi m} + O(1/m^{3/2})$.

Bernstein polynomial bases $\phi_j^m = \binom{m}{j} s^j (1-s)^{m-j}$ are better, with Lebesgue constant 1. But 2 is good.

The proof is very pretty.

One first proves by a contour integral that

$$L_{m,m}(s) - L_{m-1,m-1}(s) = \frac{1}{m+1}\binom{2m}{m}s^m(1-s)^m\,. \tag{5}$$

Then the Lebesgue function can be written

$$L_{m,m}(s) = \sum_{j=0}^{m}\frac{1}{j+1}\binom{2j}{j}s^j(1-s)^j \tag{6}$$

and this is, with $x = s(1-s)$, a truncation of the ordinary generating function for **Catalan numbers**. It is maximal when $s = 1/2$ and bounded above by 2.

Then we translate this result to the interval $-1 \le t \le 1$ for "fair" comparison to other bases. Details in the Maple Transactions paper [4].

Unbalanced blends are bad, though

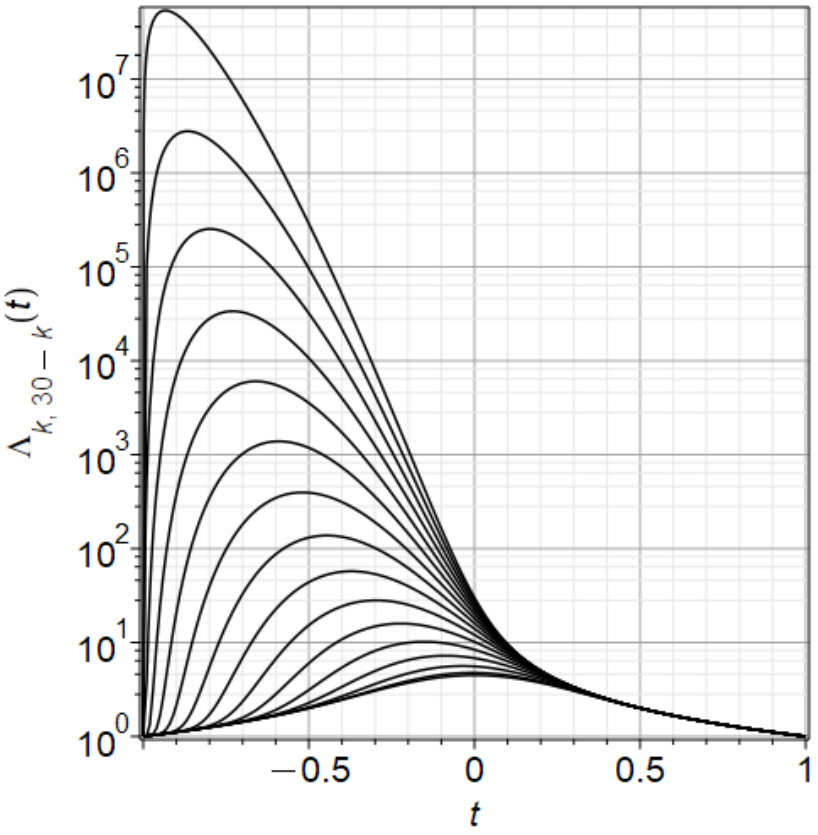

Figure 3: The Lebesgue function of *unbalanced* blends on $[-1,1]$ grows exponentially. Just as bad as equally-spaced interpolation nodes, really.

Making it jump

If we put the series at the left to be $1,0,0,\ldots$ and the series at the right to be $-1,0,0,\ldots$, no analytic function can do this. Truncating, say with $m = 368$ and $n = 631$ (so the grade $m + n + 1 = 1000$) we get a polynomial, which plots below.

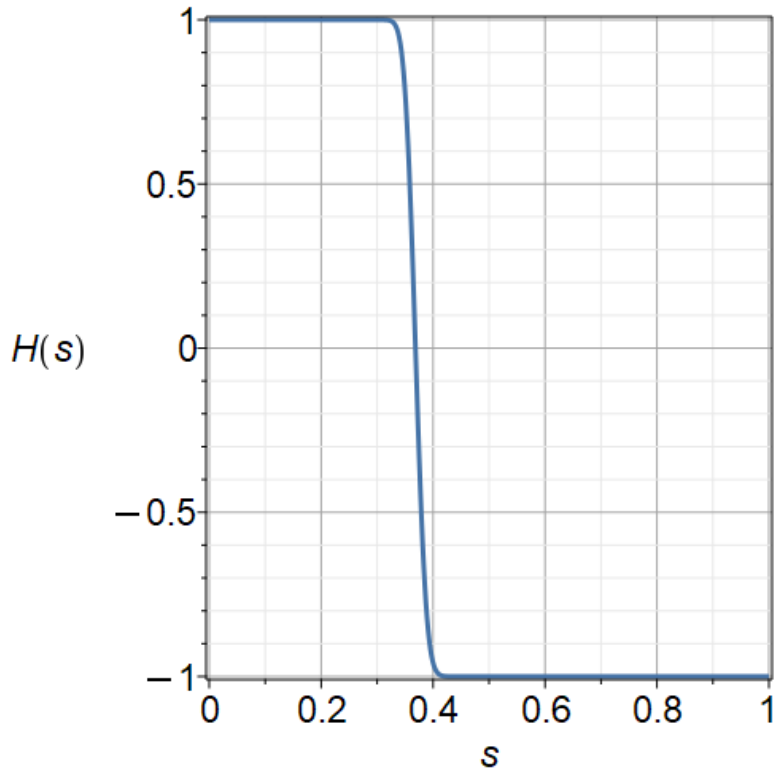

Figure 4: Look how smooth the plot is. No "Gibbs phenomenon" at all!

Blendstrings

A string of blends, or `blendstring`, is a set of the form

$$\mathcal{B} := \{L_k\}_{k=0}^{M} \tag{7}$$

where each L_k is a list of the form

$$L_k := [\alpha_k, C_{k,0}, C_{k,1}, \ldots, C_{k,m_k}] \tag{8}$$

intended to represent the known Taylor coefficients $C_{k,j}$ at the point $z = \alpha_k$.

Two blendstrings are *compatible* if they have the same knots in the same order and with the same degrees m_k at each knot. Then they can be added together, etc. If A and B are compatible blendstrings, then A op B can be a blendstring, where "op" is any of $+$, $-$, $*$, $/$ (if B is not zero), or even $\hat{}$.

A possible blendstring

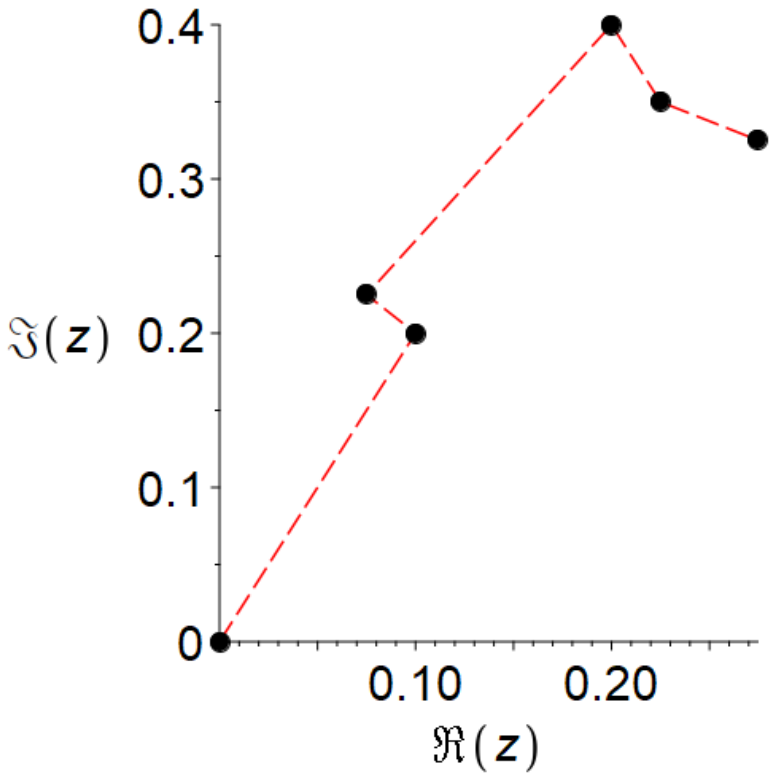

Figure 5: The knots α_k are plotted with solid circles. Taylor coefficients are known at those knots. On the segment between any two knots, Hermite's formula gives a "blend" approximating the underlying function.

Quadrature

There is a very interesting (exact!) quadrature formula for a blend, which allows *indefinite* integration of blendstrings. Obreshkov (1940) had a version of it written with ratios of binomial coefficients[4].

$$\int_{s=0}^{1} H(s)\,ds = \frac{(m+1)!}{(m+n+2)!}\sum_{j=0}^{m}\frac{(n+m-j+1)!}{(m-j)!(j+1)}p_j + \frac{(n+1)!}{(m+n+2)!}\sum_{j=0}^{n}\frac{(-1)^j(n+m-j+1)!}{(n-j)!(j+1)}q_j \qquad (9)$$

There is a five-line Maple proof of this formula. [MapleProofHermiteQuadrature.mw] Or you can use contour integration.

[4] It *might* have been a reinvention. Hermite must have known it, surely? Or Darboux or Gauss.

Indefinite integration?

The definite integral (9) from the last slide 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 $s = 1$:

$$I'(s) = H(s) = \sum_{j=0}^{m} q_j (s-1)^j$$

$$I(s) = I(1) + \sum_{j=0}^{m} \frac{q_j}{j+1} (s-1)^{j+1}\,. \tag{10}$$

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

Therefore, we can immediately find a blend for $I(s)$ on $0 \le s \le 1$.

By propagating information from the previous subinterval, this gives us a blendstring for the integral of the function being approximated by the blendstring.

Computational Version (new for this talk)

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

$$c_j = \frac{j(m-j+1)}{(j+1)(m+n+2-j)} c_{j-1} \qquad \text{for} \qquad 1 \le j \le m \tag{11}$$

$$d_j = -\frac{j(n-j+1)}{(j+1)(m+n+2-j)} d_{j-1} \qquad \text{for} \qquad 1 \le j \le n\,. \tag{12}$$

(Notice the sign alternation in the d_js.) Then

$$I(1) = \sum_{j=0}^{m} c_j p_j + \sum_{j=0}^{n} d_j q_j\,. \tag{13}$$

Continued

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

$$I(1) = \sum_{j=0}^{m} c_j \left(p_j + (-1)^j q_j\right) \tag{14}$$
$$= \frac{1}{2}(p_0 + q_0) + \frac{m}{4(2m+1)}(p_1 - q_1) + \cdots + c_m\left(p_m + (-1)^m q_m\right) .$$

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

It 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.

Differentiation

The numerical routine Erik Postma and I wrote to evaluate blends also has the ability to evaluate arbitrary derivatives, by what I call "semi-automatic differentiation".

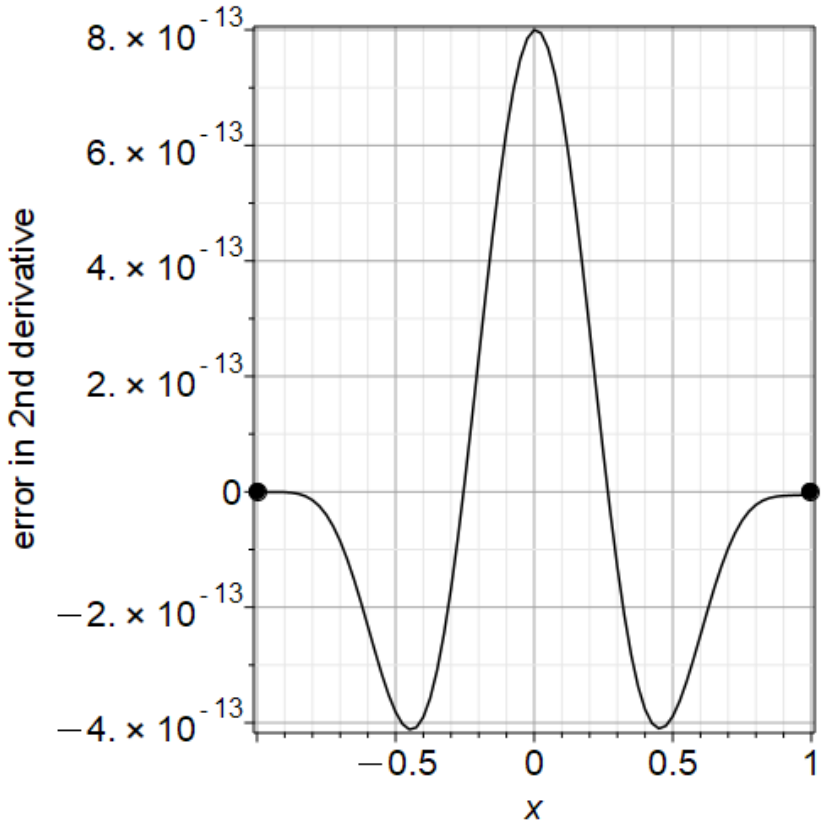

Figure 6: Error in 2nd derivative with $m = 7$ at either end

In summary

With blendstrings we can

- Approximate smooth functions on an interval in a perfectly stable, decently conditioned, and a (nearly?) spectrally efficient manner.
- Combine functions very efficiently: $f \pm g$, $f \cdot g$, f^g, and if $g \neq 0$ we can find f/g.
- Differentiate and integrate the functions, because these operations are closed on sets of compatible blendstrings.
- Find zeros.
- Solve differential equations.

Therefore, as one can with Chebfun and Approxfun, one can do a kind of symbolic computation with smooth functions, sometimes *much* more rapidly than one can do exact computation.

Where next?

I would like to add Laurent series and Puiseux series, in order to explore the singularity detection and location facilities afforded by these interpolational polynomials.

I want to add *two-sided* blendstrings, so as to allow for the possibility of jump discontinuities at the knots.

I would like to add approximate *composition* (and *decomposition*) to the things one can do with blendstrings. Seems complicated, but interesting. [Link] Some recent work of Erik Postma

I promised to try to solve delay DE with blendstrings. That should be a simple job[5]. (Once we can handle discontinuities.)

Rootfinding is important. What's the best method to find the roots of a blendstring? (J.P. Boyd & L.N. Trefethen subdivision?)

[5] "There's no such thing as a simple job."—Tim Daly

Where next, for Nick?

I wish him a happy and productive retirement!

Thank you for listening.

This work supported by NSERC, by the Spanish MICINN, and by the Isaac Newton Institute in Cambridge.

I thank Michael Monagan for comments and Ned Nedialkov for the same and for sending me a copy of Obreshkov's 1940 paper. I also thank Erik Postma and my other co-authors, and especially I thank John C. Butcher for teaching me the contour integral technique for interpolation, by which I (re)derived all these formulae.