Shape preserving interpolation with arbitrary smoothness based on mollification

Michele Guerini Rocco[a,*], Federico Scioscioli[a], Lorenzo Figini[b]

[a]Department of Physics, University of Milano-Bicocca, Milan, 20126, Italy
[b]Institute for Plasma Science and Technology, National Research Council, Milan, 20125, Italy

Abstract

A common problem in Physics and Engineering is to interpolate discrete samples while preserving the overall shape of the data: for example, maintaining positivity for a temperature curve, or monotonicity for a cumulative probability. The well-known methods do not have this property, and while several solutions can be found in the literature, they may not be smooth enough for some applications, too computationally expensive or difficult to implement. In this paper we present a shape preserving interpolation scheme that yields interpolants with an arbitrary differentiability class. It is suitable for modelling data in differential equations, fast and easy to implement. Our method is based on a convolutional operation called *mollification*, which we apply to a simpler shape preserving interpolant to achieve the desired degree of smoothness. With an appropriate polynomial mollification, the convolution has a closed form, so it can be computed quickly, giving a C^n interpolant. We define the general scheme for an arbitrary mollifier, provide an error analysis and an explicit C^2 implementation. Finally, we show how the latter was applied successfully to model the plasma density in the electron cyclotron waves raytracer GRAY.

Keywords: Smooth interpolation, Monotonicity, C^n continuity, Differential equations

1. Introduction

When modelling physical quantities in computer simulations based on data from measurements, an interpolation is often necessary. In addition to constructing a smooth function, we may need to preserve the shape of the data, by which we mean the sign for a certain number of derivatives. For example, to avoid unphysical results, the interpolation of some temperature readings should be positive everywhere; for a cumulative probability it should be everywhere nondecreasing, so that its derivative gives a distribution function.

The most common methods, splines and Hermite interpolation, are not generally shape preserving: for example, a cubic spline that interpolates positive data can easily undershoot to negative values if the samples are close to zero. Various interpolation schemes to solve this problem have been proposed over the years [1–4]. We found, however, that none were suitable for our use case: constructing data for a system of ODEs to be solved with a high-order integrator. This requires at least C^2 continuity, including when extrapolating past the boundaries. (More details and a comparison of the schemes will follow in sec. 4)

To address these shortcomings, we developed a new interpolation scheme making use of a technique from functional analysis called *mollification*. We start with a piecewise linear interpolation of the data, which by construction preserves the shape of the data but is obviously not smooth enough. However, by applying a convolution with a *mollifier* – a smooth function with compact support – the initial rough interpolant becomes smooth while maintaining its shape preserving properties. In particular, the mollified interpolant inherits the differentiability class of the mollifier.

The use of convolutions for smoothing data is standard, for example, in signal processing, where several filters, notably finite impulse reponse (FIR) filters, can be interpreted as discrete convolutions. Our method, however, is based on a continuous convolution and is more similar, in this respect, to kernel methods in nonparametric statistics. The choice of mollifiers – here made practical by relaxing the standard definition from C^∞ functions to polynomials with C^n differentiability at the boundary – enables the construction of interpolants with a simple closed form and, critically, gives the required properties of continuity, shape preservation and compact support.

Unlike Hermite interpolation, but similar to smoothing splines, this method is not strictly interpolating: the distance between the interpolant and the samples, the error, is non-zero but can be made arbitrarily small by adjusting the width of the mollifier. As a side effect, mollification tends to smooth out noise by effectively acting as a low-pass filter. This can be useful when the data are the result of an uncertain measurement and have not been processed before interpolating.

In the next section we give a brief introduction to mollifiers and prove their main properties. All the results in this section are well known, though typically only stated in the C^∞ case. The smoothness and shape preservation of our interpolation scheme follow immediately from these results. In sec. 3 we give the definition of the scheme for a generic mollifier with arbitrary differentiability class, an error analysis and an explicit implementation with a C^2 mollifier. Finally, in sec. 4 we discuss other shape preserving methods and show how they com-

*Corresponding author.
Email address: michele.guerinirocco@unimib.it (Michele Guerini Rocco)

pare to our own.

2. Mollifier theory

We define a mollifier as a continuous function $g : \mathbb{R}^d \to \mathbb{C}$ with the following properties[5]:

1. $\text{supp}(g) = \overline{\{x \in \mathbb{R}^d \mid g(x) \neq 0\}}$ is compact
2. $\int g(x)dx = 1$
3. $\lim_{w\to 0^+} w^{-d} \int g\left((x_0 - x)/w\right) f(x)dx = f(x_0)$ for almost all x_0 and for all $f \in L^1_{\text{loc}}(\mathbb{R}^d)$

Consider the space $D \equiv C^n_c(\mathbb{R}^d)$ of C^n functions with compact support[1]. We take a $g \in D$ (implying prop. 1) and, if necessary, normalize it such that prop. 2 holds as well; then we define $g_w(x) \equiv g(x/w)w^{-d}$ for $w > 0$, the width of the mollifier. To prove g is a mollifier,

Theorem 1 (identity) $\lim_{w\to 0} \int g_w(x_0 - x)f(x)dx = f(x_0)$, almost everywhere. Or, $g_w \to \delta$ in the Schwartz distribution sense, where δ is the Dirac δ distribution.

Proof. Using prop. 2 we can rewrite

$$\begin{aligned}\int g_w(x_0 - x)f(x)dx - f(x_0) &= \int g_w(x_0 - x)f(x)dx \\ &\quad - \int g_w(x_0 - x)dx f(x_0) \\ &= \int g_w(x_0 - x)[f(x) - f(x_0)]dx,\end{aligned}$$

so we have:

$$\left|\int g_w(x_0 - x)f(x)dx - f(x_0)\right| \leq \int |g_w(x_0-x)|\,|f(x)-f(x_0)|\,dx.$$

By prop. 1 $\text{supp}(g_w) \subseteq B^d(0, cw)$, for some $c > 0$. Then

$$\int |g_w(x_0-x)|\,|f(x)-f(x_0)|\,dx \leq \frac{C}{w^d}\int_{B^d(x_0,cw)} |f(x)-f(x_0)|\,dx$$

for some constant $C > 0$, since $g(x/w)$ is bounded. For $w \to 0$ the limit vanishes a.e. by the Lebesgue differentiation theorem. □

We write $f * g_w \to f$, where $*$ stands for convolution. The function $f_w = f * g_w$ is what is called the mollification of f, more explicitly:

$$f_w(x_0) = \int f(x)g((x_0 - x)/w)w^{-d}dx$$

We now show that the mollification is C^n.

Lemma 1 (continuity) f_w is (uniformly) continuous.

Proof.

$$\begin{aligned}|f_w(x_0') - f_w(x_0)| &= \left|\int f(x)[g_w(x_0' - x) - g_w(x_0 - x)]dx\right| \\ &\leq \int_K |f(x)|\,\left|g_w(x_0' - x) - g_w(x_0 - x)\right|\,dx\end{aligned}$$

where K is the closure of a ball enclosing both $x_0 + \text{supp}\, g_w$ and $x_0' + \text{supp}\, g_w$. By the mean value theorem for g_w:

$$|f_w(x_0') - f_w(x_0)| \leq \int_K |f(x)|dx\,|x_0' - x_0|\,\max|\nabla g_w| \to 0$$

uniformly as $x_0' \to x_0$. □

For the existence of the partial derivatives,

Lemma 2 (derivatives) $f_w \in C^n(\mathbb{R}^d)$ for all $w > 0$.

Proof.

$$\begin{aligned}\nabla_i f_w(x_0) &= \lim_{\varepsilon\to 0} \frac{f_w(x_0 + \varepsilon e_i) - f_w(x_0)}{\varepsilon} \\ &= \lim_{\varepsilon\to 0} \int f(x) \frac{g_w(x_0 - x + \varepsilon e_i) - g_w(x_0 - x)}{\varepsilon} dx\end{aligned}$$

For a fixed x and $k > 0$ denote

$$\Delta_k(x_0) \equiv f(x)\left[g_w(x_0 - x + e_i/k) - g_w(x_0 - x)\right]k$$

By the mean value theorem, $\Delta_k(x_0) = f(x)\,\nabla_i g_w(x_1 - x)$ for some $x_1 \in B(x_0, 1/k)$. So the sequence $|\Delta_k(x)|$ is bounded by

$$m(x) = \max_{x_1} |f(x)\,\nabla_i g(x_1 - x)|$$

Since $g_w \in D$ we also have $\int m(x)dx < +\infty$, so by the Lebegue dominated convergence we conclude that

$$\nabla_i f_w(x_0) = \int f(x)\,\nabla_i g_w(x_0 - x)dx$$

The existence of higher-order partial derivatives follows by induction. Note: if $f \in C^m$ already, we could also prove $f_w \in C^{m+n}$ by switching the roles of f and g_w for the first m derivatives. □

We prove a final theorem that will be useful to give an error bound on the interpolation:

Theorem 2 (convergence) If $f \in C^0(U)$, on an open $U \subseteq \mathbb{R}^d$, then the mollification $f_w \to f$ uniformly for $w \to 0$ on any compact subset $K \subseteq U$.

Proof. By the Heine-Cantor theorem f is uniformly continuous in K:

$$\forall \varepsilon > 0 : \exists \delta > 0 : \forall x, y \in K : |x - y| < \delta \Rightarrow |f(x) - f(y)| < \varepsilon$$

So, for $x_0 \in K$ and $w < \delta$

$$\begin{aligned}|f_w(x_0) - f(x_0)| &\leq \int |f(x) - f(x_0)|\,|g_w(x_0 - x)|\,dx \\ &< \varepsilon \int |g_w(x_0 - x)|dx = \varepsilon C\end{aligned}$$

□

[1] Normally mollifiers are defined for smooth, C^∞, functions, but in the present article we relax this constraint.

3. Mollifier interpolation scheme

3.1. Definition

We consider a set of samples $(x_i, y_i) \in \mathbb{R}^2$ for $i = 1...N$ such that $x_1 < x_2 < ... < x_N$ and define the piecewise linear interpolation f of these samples:

$$f(x) = \sum_{i=1}^{N-1} (m_i x + q_i)\chi_{U_i}(x) \tag{1}$$

where:

- $m_i \equiv \Delta_i/h_i = (y_{i+1} - y_i)/(x_{i+1} - x_i)$
- $q_i = y_i - m_i x_i$
- $U_i = [x_i, x_{i+1})$ if $i < N-1$, $[x_i, x_{i+1}]$ otherwise
- χ is the characteristic function.

Now we mollify f using a positive mollifier g. For simplicity, we assume $\mathrm{supp}(g) = [-1, 1]$. The convolution integral splits over the $N-1$ intervals U_i:

$$\begin{aligned} f_w(x_0) &= \sum_{i=1}^{N-1} \int_{U_i} (m_i x + q_i) g((x_0 - x)/w)/w \, dx \\ &= \sum_{i=1}^{N-1} \int_{V_i(x_0)} (m_i x_0 - m_i w t + q_i) g(t) dt \end{aligned}$$

where $V_i(x_0) = (x_0 - U_i)/w$. So the mollification reduces to a sum of just two kinds of integrals:

$$f_w(x_0) = \sum_{i=1}^{N-1} [(m_i x_0 + q_i) J_i(x_0) - m_i w K_i(x_0)] \tag{2}$$

with $J_i(x_0) = \int_{V_i(x_0)} g(t)dt$ and $K_i(x_0) = \int_{V_i(x_0)} t g(t)dt$.

We can directly differentiate eq. 2 to obtain expressions for the derivatives of the interpolant. Equivalently, we can mollify the distributional derivative of f: Df will be a step function, $D^2 f$ a linear combination of δ distributions, etc. Either way, for $x \in [x_1, x_N]$ we find[2]:

$$\begin{aligned} Df_w(x_0) &= \sum_{i=1}^{N-1} m_i J_i(x_0) \\ D^2 f_w(x_0) &= \frac{1}{w} \sum_{i=1}^{N-1} (m_{i+1} - m_i) g\left(\frac{x_0 - x_i}{w}\right) \\ D^3 f_w(x_0) &= \frac{1}{w^2} \sum_{i=1}^{N-1} (m_{i+1} - m_i) Dg\left(\frac{x_0 - x_i}{w}\right) \\ &... \end{aligned} \tag{3}$$

Eqns. 2, 3 constitutes our explicit solution to the problem of shape preserving interpolation with an arbitrary smoothness requirement. We make a few observations:

1. Mollification enlarges the support of f from $[x_1, x_N]$ to $[x_1 - w, x_N + w]$, so if fixing the endpoint values is critical, the abscissa must be rescaled accordingly;
2. The integrals J_i, K_i depends on x_0 through V_i, so they must be computed on every evaluation.
3. However, most terms in the summation will be zero unless w is very large: only those for which $|x_0 - x_i| \leq w$ contribute.

3.1.1. Shape preservation

To prove that the method is shape preserving we first note that eq. 1 locally maintains the sign of the data y_i and the first finite differences $\Delta_i = y_{i+1} - y_i$. In fact, assuming y_j, y_{j+1} have same signs, for any $x_0 \in U_j$ we see:

1. $\mathrm{sign}\, f(x_0) = \mathrm{sign}(y_j(1-\eta) + \eta y_{j+1}) = \mathrm{sign}\, y_j$, since $\eta = (x_0 - x_j)/h_i$ is such that $0 \leq \eta \leq 1$.
2. $\mathrm{sign}\, Df(x_0) = \mathrm{sign}(\Delta_j/h_j) = \mathrm{sign}\, \Delta_j$, since $h_j > 0$

We now rewrite the definition of f_w as

$$f_w(x_0) = \int_{-1}^{+1} f(x_0 - tw) g(t)\, dt.$$

Since g is positive, the sign of the integral is solely determined by $f(x_0 - tw)$. Assuming all samples x_k such that $|x_k - x_0| \leq w$ have the same sign, for example $y_k \geq \varepsilon > 0$, we have

$$f_w(x_0) \geq \varepsilon \int_{-1}^{+1} g(t)\, dt = \varepsilon > 0$$

proving that f_w locally preserves the sign of the data.

For the derivatives, we make the similar assumption that for all k such that $|x_k - x_0| \leq w$, the finite differences $\Delta_k = y_{k+1} - y_k$ and $\Delta_k^2 = \Delta_{k+1} h_k/h_{k+1} - \Delta_k$ have definite sign. Then, by noting that all $J_k(x_0)$ are positive, it follows from eq. 3 that:

1. $\mathrm{sign}\, Df_w(x_0) = \mathrm{sign}\, \Delta_k$,
2. $\mathrm{sign}\, D^2 f_w(x_0) = \mathrm{sign}\, \Delta_k^2$.

For the n-th derivative with $n > 2$, instead, the method fails to preserve the signs, in general. This can be seen from the fact that the terms $D^{n-2} g$ entering eq. 3 can change sign. So, we conclude that the method is shape preserving up to the second derivative.

3.2. Error analysis

As anticipated, we now provide an error bound for the interpolant eq. 2 making use of the uniform convergence of $f_w \to f$ (theorem 2). Following the same reasoning as before, but substituting eq. 1 for f, we find explicitly:

$$\begin{aligned} f_w(x_0) - f(x_0) &= \int [f(x) - f(x_0)] g_w(x_0 - x) dx \\ &= \sum_{i=1}^{N-1} \int_{U_i} m_i \frac{x - x_0}{w} g\left(\frac{x_0 - x}{w}\right) dx \\ &= \sum_{i=1}^{N-1} (-m_i w) K_i(x_0) \end{aligned}$$

[2] Technically eq. 1 prescribes two jumps at $x = x_1, x_N$ that introduce additional terms in the derivatives of order ≥ 2; however we avoid them by extrapolating with constant y_1, y_N outside the interpolation interval.

The error bound becomes: $\varepsilon_w = 35/128\ w \max_i |m_i|$.
For our use case, extrapolation past the boundaries was required: this can be done in a variety of ways, the simplest of which is extending f with a fixed value. In eq. 2 this amounts to setting $m_0 = m_N = 0$, $q_0 = y_1, q_N = y_N$ and

$$V_0 = [\max((x_0 - x_1)/w), -1),\ 1)$$
$$V_N = [-1,\ \min((x_0 - x_N)/w), +1)]$$

A Python/NumPy implementation can be found in app. A.

4. Comparison with existing schemes

We give a brief presentation of some of the shape preserving interpolation schemes found in the literature and how they compare to our proposed one.

A well-known extension to the standard Hermite cubic interpolation is the Fritsch-Carlson algorithm [1]. This preserves the monotonicity by adjusting the estimated derivatives of the data. As Hermite interpolation is local, this method can be applied to piecewise monotonic data as well as global. However, this scheme only guarantees C^1 continuity and is not easily extended to become C^2.

A monotonicity-preserving C^2 spline method was described by Delbourgo and Gregory [2] using quadratic rational functions. This method assumes globally and strictly monotonic data, unfortunately not valid in our scenario. The condition can be relaxed by splitting the interpolation range into several subintervals where the data are either monotonic or constant; but since the 2nd derivatives at the boundaries are constrained, doing so reduces the differentiability class to C^1.

An evolution of the Delbourgo and Gregory method are rational *cubic* splines. These have been developed extensively by Karim and Kong [3,6]: their method is C^2 and allows for finer control over the shape of the interpolant by adding three free parameters. Potentially, the 2nd derivative at the boundary could also be controlled using some of these extra parameters, however – as presented by Karim and Kong – the scheme suffers from the same limitation as that of Delbourgo and Gregory. Another downside of this method is that enforcing monotonicity – for example – requires multiple iterations of adjusting the shape, recomputing all the derivatives at the knots, re-adjusting the shape... so it can be very expensive.

Finally, we consider smoothing splines or P-splines. These are B-splines obtained as the solution of a least squares problem with a penalty – hence the name – on higher order derivatives, thus producing less noisy interpolants. Unlike the previous schemes, but like ours, these are not strictly interpolating: the error however is easily controlled by adjusting the weight of the penalty and this can also be used to model noisy data. By adding an asymmetric penalty – promoting a given sign of a derivative – P-splines can be made shape-preserving. A simple algorithm for a monotonic P-spline has been proposed by Eilers in [4]. Some downsides of this scheme are that it is approximate – small monotonicity violations are still possible – and that there are a greater number of parameters to be determined: the position and number of knots, the weights of the penalties for smoothness λ and monotonicity κ.

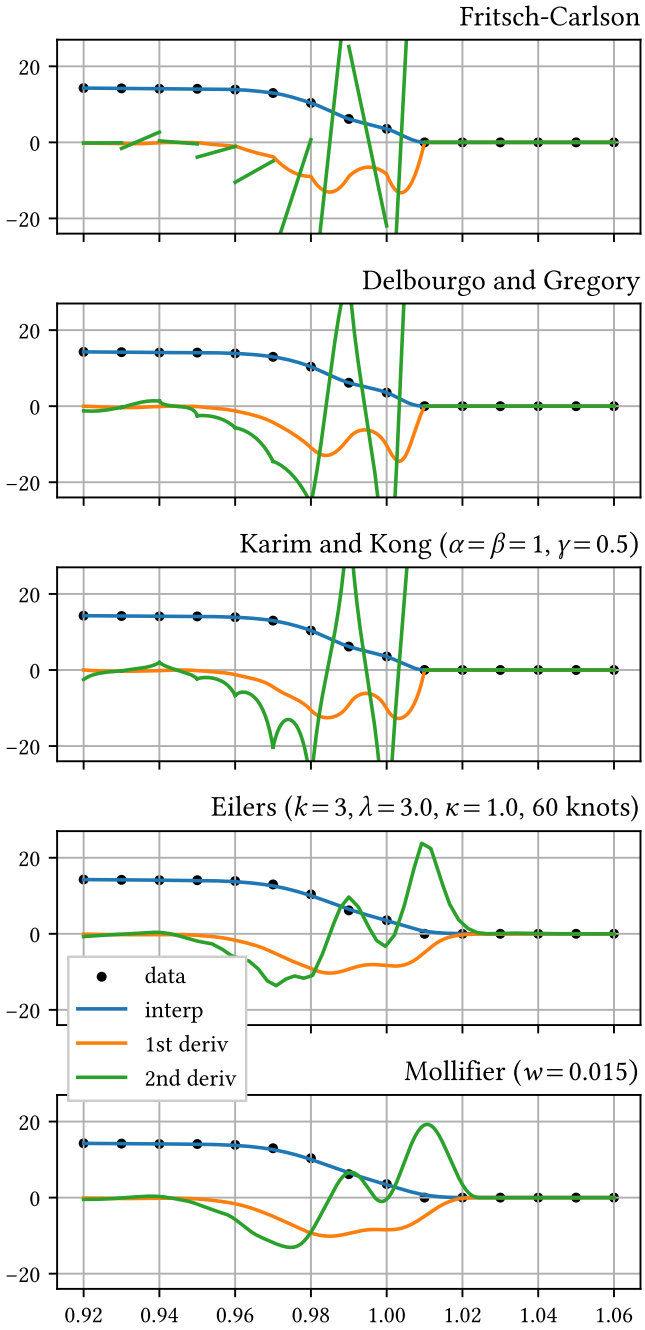


Figure 2: Comparison of interpolants and their first two derivatives for the mentioned schemes. The dataset, reported in tbl. 1, is taken from the edge of a plasma density radial profile. The y scale of the derivatives is the same in each subplot. Code to generate the plots can be found in the supplementary material.

Table 1: Example plasma density profile near the boundary. This is a portion of the density radial profile for the plasma scenario "E1" of the Divertor Test Tokamak (DTT) [7]. x is the normalised toroidal radius and y the electron number density measured in $10^{20}m^{-3}$.

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
x	0.92	0.93	0.94	0.95	0.96	0.97	0.98	0.99	1.00	1.01	1.02	1.03	1.04	1.05	1.06
y	14.3	14.2	14.1	14.0	13.9	13.0	10.4	6.2	3.6	0.0	0.0	0.0	0.0	0.0	0.0

4.1. Numerical demonstrations

We now compare the interpolation schemes applied to data relevant for magnetically confined plasma simulations. GRAY [8] is an electron cyclotron waves raytracer, used to model electron cyclotron resonant heating (ECRH) in tokamaks. GRAY propagates a Gaussian beam as a discrete number of rays by numerically solving a set of 2nd order coupled ODEs. The integration scheme requires all physical quantities to be at least C^2 continuous on a ray; in particular the plasma density should be smooth at the plasma boundary and vanish in the vacuum outside, where the beam originates. Typically the plasma density is provided as a set of (x, y) samples, where x is a normalised radial variable. Naturally, the density is positive and its profile is expected to decrease monotonically from the boundary ($x = 1$) to exactly 0 in the vacuum (at some $x > 1$).

In fig. 2 we see a comparison of all five methods on a typical plasma density profile (tbl. 1). While all interpolants are visually similar, their derivatives behave very differently. The Fritsch-Carlson algorithm, being a cubic interpolation, has a piecewise linear 2nd derivative with large jumps. The rational spline methods give close results: the first derivatives are similar in shape to that of Fritsch-Carlson but smoother; the 2nd derivatives are continuous but very irregular. As explained, both methods require restricting the domain to $x \leq 1.01$ to ensure the data are strictly monotonic, but the 2nd derivatives then become significantly discontinuous at $x = 1.01$.

Lastly, Eilers and our method give very similar results: both are C^2 with smaller and smoother derivatives than the previous ones and are reasonably close to the data. In fact, if we compute the quadratic error

$$r = \sum_{i=1}^{n} (f(x_i) - y_i)^2,$$

we find r=0.89 for the P-spline and r=0.93 for the mollification. Our method, however, produces an even smoother 2nd derivative and is exactly zero for $x \geq 1 + w$; the P-spline instead only decays approximately as an exponential (see fig. 3) which is problematic for our use case. In fact, GRAY assumes a perfect vacuum outside the plasma boundary and a non-zero density – albeit very small – would either make the simulation numerically inconsistent or require to solve a complex dispersion relation in order to obtain the initial conditions of the rays.

We must note that mollification gives good results as long as the width w is not significantly smaller than the distance between two samples. In this condition, as can be seen in fig. 4, it essentially reduces to a linear interpolation with smooth joints. In comparison, Eilers method produces less artificial shapes even when the number of knots is larger than the number of samples. In our use case, however, the resolution is high enough that this is not a concern.

4.2. Benchmark

The mollification scheme is very computationally efficient. To show this, we performed a benchmark against an implementation of Eilers method based on the `BSpline` class from Scipy [9]. To obtain fairer results – considering that SciPy uses native code – we applied the Numba JIT compiler [10] to our pure Python code.

We then measured the execution time of computing and separately evaluating the interpolants (using the same dataset from tbl. 1) on 1000 equispaced points. To minimise the noise in the execution environment, the tests were repeated 500000 times and the minimum time was taken as the final value. The results are: it takes 166 µs to obtain the P-spline coefficients and 36 µs to evaluate it; the mollifier does not require any computation ahead of evaluation and takes 17 µs to evaluate.

In conclusion, the computational cost of our scheme is comparable to the evaluation of a cubic spline, with the benefit of having no extra initialisation costs.

5. Conclusions

We have presented a new interpolation scheme based on mollification that is shape-preserving – meaning it maintains the sign, monotonicity and convexity of the data – and can be made arbitrarily smooth. This is well-suited to physical simulations and specifically to construct data for differential equations, where functions with C^n continuity and compact support may be desired.

We have provided a C^2 implementation using a 6th degree polynomial mollifier, shown that it is practicable to implement and that it gives similar results to smooth spline methods. In addition, the scheme is computationally efficient: comparable to the evaluation of a cubic spline.

6. Acknowledgements

We would like to thank prof. Alberto Zaffaroni for its help in reviewing our proofs.

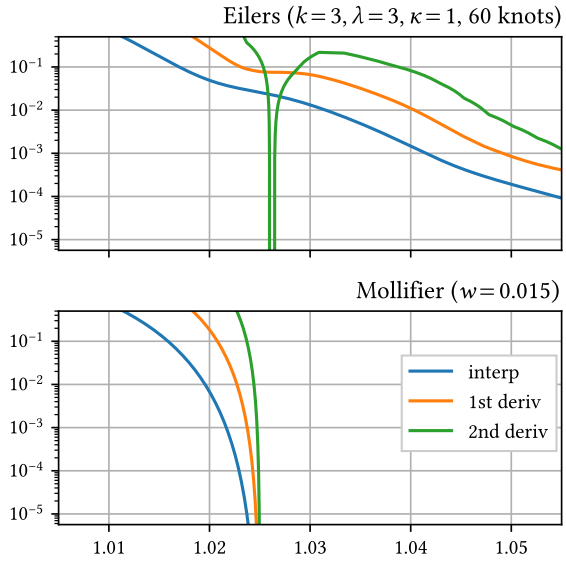


Figure 3: Semilog plot showing the decay to zero in the region $x > 1$ of the same interpolants from fig. 2. The P-spline and derivatives decreases approximately exponentially (in absolute value), but are everywhere positive; the mollification vanishes and stays constant.

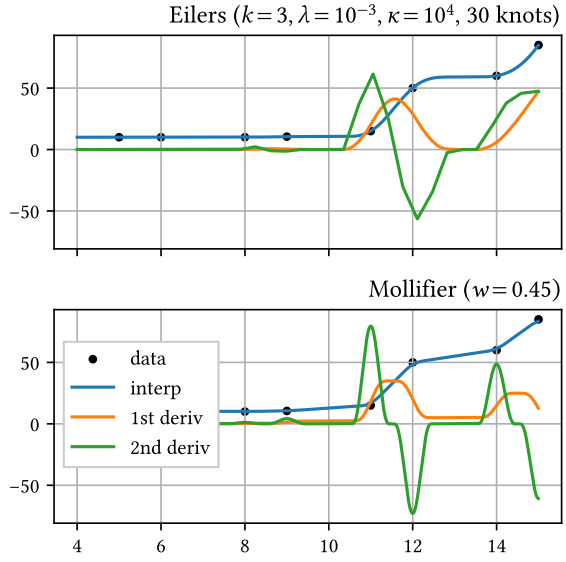


Figure 4: Comparison between Eilers and our scheme in a case with sparse sample points, compared to the mollifier width. The dataset is an example to test for monotonicity that can be found in [2].

CRediT authorship contribution statement

M. Guerini Rocco: Conceptualization, Formal analysis, Methodology, Investigation, Software, Writing – original draft, Writing – review and editing, Visualization, Validation, Supervision. **F. Scioscioli**: Writing – review and editing, Validation. **L. Figini**: Writing – review and editing, Validation, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The example datasets, implementations of the rational splines and P-spline methods, along with code to reproduce the benchmark and to generate the figures is available at https://doi.org/10.17632/dsfjwzsdhv.1.

Appendix A. Python implementation

Here follows a Python/NumPy implementation of the C^2 mollifier interpolation scheme:

```
from typing import Callable
import numpy as np

def g(x: float) -> float:
    if abs(x) >= 1:
        return 0.0
    return 70 * (1/2 + x/2)**3 * (1/2 - x/2)**3

def g_integral(x: float) -> float:
    """
    Returns the integral ∫₀^x g(t) dt
    """
    return x*(x**2*(x**2*(21 - 5*x**2) - 35) + 35)/32

def tg_integral(x: float) -> float:
    """
    Returns the integral ∫₀^x t·g(t) dt
    """
    return -35*x**2*(x**2 - 2)*(x**2*(x**2 - 2) + 2)/256

def interpolate(x: list[float], y: list[float],
                w: float) -> Callable[[float, int], float]:
    """
    Returns the mollification
      f_w(x₀) = ∫ f(x)·g((x₀-x)/w)/w dx;
    where f is a linear interpolation of the data (x, y),
          g is a C² mollifier.
    """
    def f_w(x0: float, deriv: int) -> float:
        # Find points within [x₀-w, x₀+w]
        left = np.searchsorted(x, x0-w, side='right')
        right = np.searchsorted(x, x0+w, side='left')
        f_w = 0
        for i in range(left, right + 1):
            if i == 0:
                # y=y[0] for x≤x[0]
                m = 0
                q = y[0]
                a = max((x0-x[i])/w, -1)
                b = 1
            elif i == x.size:
                if deriv == 2:
                    break
```

```
                # y=[n] for x≥x[n]
                m = 0
                q = y[-1]
                a = -1
                b = min((x0-x[-1])/w, +1)
            else:
                m = (y[i] - y[i-1]) / (x[i] - x[i-1])
                q = y[i-1] - m * x[i-1]
                a = max((x0-x[i])/w, -1)
                b = min((x0-x[i-1])/w, +1)

            if deriv == 0:
                # Compute f*g
                J = g_integral(b) - g_integral(a)
                K = tg_integral(b) - tg_integral(a)
                f_w += (m*x0+q)*J - (m*w)*K
            elif deriv == 1:
                # Compute Df*g
                J = g_integral(b) - g_integral(a)
                f_w += m*J
            elif deriv == 2:
                # Compute D²f*g
                if i == x.size - 1:
                    m1 = 0
                else:
                    m1 = (y[i+1] - y[i]) / (x[i+1] - x[i])
                f_w += (m1-m) * g((x0-x[i])/w)/w
        return f_w

    return np.vectorize(f_w)
```

References

[1] F.N. Fritsch, R.E. Carlson, Monotone piecewise cubic interpolation, SIAM Journal on Numerical Analysis. 17 (1980) 238–246. https://doi.org/10.1137/0717021.

[2] R. Delbourgo, J.A. Gregory, Shape preserving piecewise rational interpolation, SIAM Journal on Scientific and Statistical Computing. 6 (1985) 967–976. https://doi.org/10.1137/0906065.

[3] S.A. Abdul Karim, K. Voon Pang, Shape preserving interpolation using C^2 rational cubic spline, Journal of Applied Mathematics. 2016 (2016) 1–14. https://doi.org/10.1155/2016/4875358.

[4] P.H.C. Eilers, Unimodal smoothing, Journal of Chemometrics. 19 (2005) 317–328. https://doi.org/10.1002/cem.935.

[5] L.C. Evans, Partial differential equations, 2nd ed., American Mathematical Society, Providence, R.I., 2010. ISBN: 9780821849743. https://www.worldcat.org/oclc/465190110.

[6] S.A. Bin Abdul Karim, Rational cubic spline interpolation for monotonic interpolating curve with C2 continuity, MATEC Web of Conferences. 131 (2017) 4016. https://doi.org/10.1051/matecconf/201713104016.

[7] I. Casiraghi, P. Mantica, F. Koechl, R. Ambrosino, B. Baiocchi, A. Castaldo, J. Citrin, M. Dicorato, L. Frassinetti, A. Mariani, P. Vincenzi, P. Agostinetti, L. Aucone, L. Balbinot, S. Ceccuzzi, L. Figini, G. Granucci, P. Innocente, T. Johnson, H. Nyström, M. Valisa, First principle-based multi-channel integrated modelling in support of the design of the divertor tokamak test facility, Nuclear Fusion. 61 (2021) 116068. https://doi.org/10.1088/1741-4326/ac21b9.

[8] D. Farina, A quasi-optical beam-tracing code for electron cyclotron absorption and current drive: GRAY, Fusion Science and Technology. 52 (2007) 154–160. https://doi.org/10.13182/fst07-a1494.

[9] P. Virtanen, et al., SciPy 1.0: Fundamental algorithms for scientific computing in python, Nature Methods. 17 (2020) 261–272. https://doi.org/10.1038/s41592-019-0686-2.

[10] S.K. Lam, A. Pitrou, S. Seibert, Numba: A LLVM-based python JIT compiler, in: Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC, ACM, 2015: pp. 1–6. https://doi.org/10.1145/2833157.2833162.