Pyramid algorithms for barycentric rational interpolation

Kai Hormann · Scott Schaefer

Abstract

We present a new perspective on the Floater–Hormann interpolant. This interpolant is rational of degree (n,d), reproduces polynomials of degree d, and has no real poles. By casting the evaluation of this interpolant as a pyramid algorithm, we first demonstrate a close relation to Neville's algorithm. We then derive an $O(nd)$ algorithm for computing the barycentric weights of the Floater–Hormann interpolant, which improves upon the original $O(nd^2)$ construction.

Citation Info

Journal
Computer Aided Geometric Design

Volume
42, February 2016

Pages
1–6

1 Introduction

Given the $n+1$ interpolation nodes $x_0 < x_1 < \cdots < x_n$ and the associated data $f_0, f_1, \ldots, f_n$, there are two ways to write the rational Floater–Hormann interpolant [3] of degree $d \le n$. On the one hand, it can be expressed as the blend

$$r(x) = \frac{\sum_{i=0}^{n-d} \lambda_i(x) p_i(x)}{\sum_{i=0}^{n-d} \lambda_i(x)} \tag{1}$$

of the polynomials p_i of degree d, which locally interpolate the data $f_i, \ldots, f_{i+d}$, with weighting functions

$$\lambda_i(x) = \frac{(-1)^i}{(x-x_i)\cdots(x-x_{i+d})}.$$

On the other hand, it can be written in the barycentric form

$$r(x) = \sum_{i=0}^{n} \frac{(-1)^i}{x-x_i} w_i f_i \Bigg/ \sum_{i=0}^{n} \frac{(-1)^i}{x-x_i} w_i \tag{2}$$

with positive weights

$$w_i = \sum_{j=\max(0,i-d)}^{\min(i,n-d)} \prod_{k=j,k\neq i}^{j+d} \frac{1}{|x_i - x_k|}. \tag{3}$$

The barycentric form is particularly suited for evaluating the interpolant r in $O(n)$ time, once the weights w_i, which depend only on the nodes x_i and not on the data f_i, have been precomputed. Using (3), these weights can be determined in $O(nd^2)$ steps, which is exactly how common libraries like *Numerical Recipes*[1] [5] and *ALGLIB* [2] perform the computation.

We present two novel procedures for evaluating $r(x)$, which are inspired by Ron Goldman's pyramid algorithms [4] for the evaluation of polynomial and spline curves. While the first is tailored for Floater–Hormann interpolants and exploits the representation of r in (1), the second is based on (2) and works for general barycentric rational interpolants of degree (n,n) with arbitrary weights w_i. Both algorithms require $O(n^2)$ operations and are closely related to Neville's algorithm for constructing interpolating polynomials of degree n. They further lead to a novel $O(nd)$ algorithm for computing the weights in (3).

[1] The claim in [5, page 128] that "the workload to construct the weights is of order $O(nd)$ operations" is wrong, because the given code just implements the formula in (3) in a straightforward way and is clearly of order $O(nd^2)$.

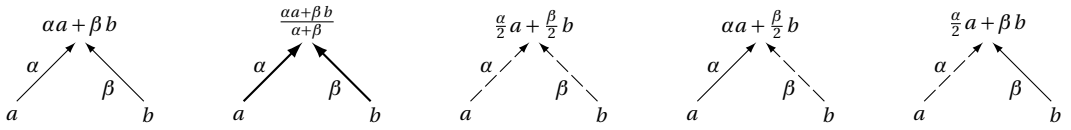

Figure 1: Pyramid notation for linear combinations. Thick arrows indicate affine combinations, where we omit the normalization factors of the weights to keep the diagram less cluttered. Dashed arrows indicate that the weights need to be multiplied with 1/2.

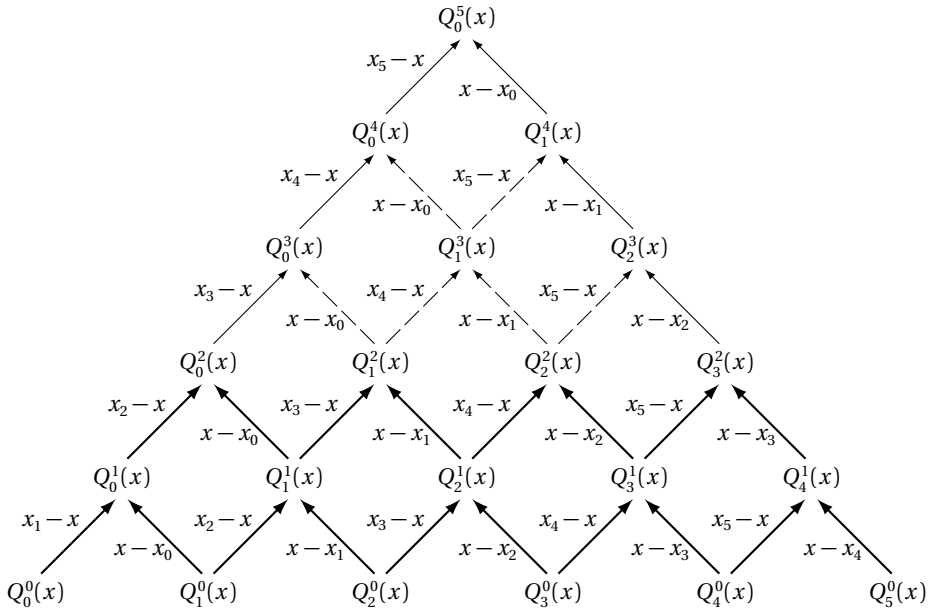

Figure 2: Example of the pyramid algorithm for Floater–Hormann interpolation with $n=5$ and $d=2$.

2 Evaluating the rational interpolant

Using the notation in Figure 1, the Floater–Hormann interpolant in (1) can be evaluated by the pyramid algorithm in Figure 2, which is a slight variation of Neville's algorithm [4, Chapter 2.2], where the weights in the top $n-d$ rows of the pyramid are not normalized and the weights at the interior edges in these rows are multiplied with an additional factor of $1/2$. That is, for some given evaluation parameter x, we start with the initial data

$$Q_i^0(x)=(f_i,1), \qquad i=0,\dots,n$$

and compute the bottom d rows of the pyramid with Neville's algorithm as

$$Q_i^\ell(x)=\frac{x_{i+\ell}-x}{x_{i+\ell}-x_i}Q_i^{\ell-1}(x)+\frac{x-x_i}{x_{i+\ell}-x_i}Q_{i+1}^{\ell-1}(x), \qquad i=0,\dots,n-\ell,$$

for $\ell=1,\dots,d$, resulting in the values

$$Q_i^d(x)=(p_i(x),1), \qquad i=0,\dots,n-d.$$

We then continue to compute the top $n-d$ rows of the pyramid as

$$Q_i^\ell(x)=\eta_i^{n-\ell+1}(x_{i+\ell}-x)Q_i^{\ell-1}(x)+\eta_{i+1}^{n-\ell+1}(x-x_i)Q_{i+1}^{\ell-1}(x), \qquad i=0,\dots,n-\ell,$$

for $\ell=d+1,\dots,n$, where

$$\eta_i^\ell=\begin{cases}1, & \text{if } i=0 \text{ or } i=\ell,\\ 1/2, & \text{otherwise,}\end{cases}$$

resulting in the value

$$Q_0^n(x)=\big(f(x),g(x)\big).$$

We now observe that the diagram in Figure 2 satisfies the *parallel property*, that is, parallel arrows all have the same labels. Hence the product of labels along any path from $Q_i^d(x)$ to the apex of the pyramid is always

$$\mu_i(x)=\prod_{j=0}^{i-1}(x-x_j)\prod_{k=i+d+1}^{n}(x_k-x)=(-1)^{n-d}\pi(x)\lambda_i(x),$$

where

$$\pi(x)=\prod_{j=0}^{n}(x-x_j).$$

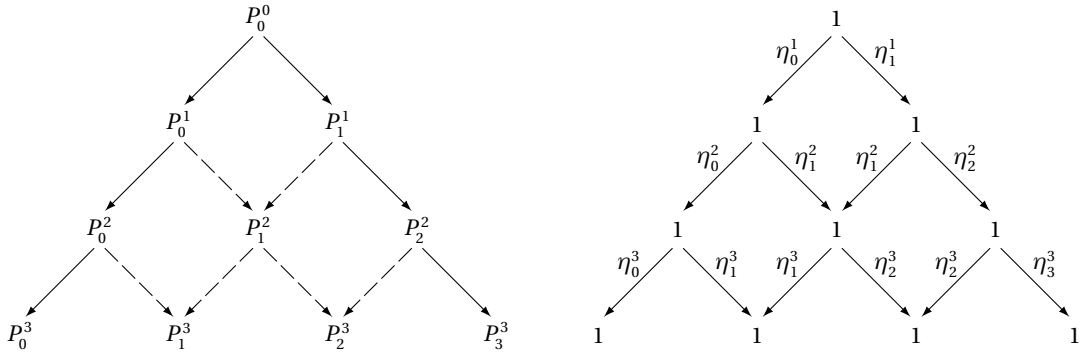

Figure 3: Recurrence of the values P_i^ℓ (left), which are all equal to 1 (right).

Due to the additional factors η_i^ℓ, each of these paths is further multiplied by some power of $1/2$. Denoting the sum of these powers by P_i^{n-d}, we have

$$Q_0^n(x) = \sum_{i=0}^{n-d} P_i^{n-d} \mu_i(x) Q_i^d(x).$$

As any path from the node $Q_i^\ell(x)$ to the apex must traverse either of the nodes $Q_{i-1}^{\ell-1}(x)$ or $Q_i^{\ell-1}(x)$, it is clear that the values P_i^ℓ satisfy the recurrence[2]

$$\begin{aligned} &P_0^0 = 1, \\ &P_i^\ell = \eta_i^\ell P_{i-1}^{\ell-1} + \eta_i^\ell P_i^{\ell-1}, \qquad i = 0,\dots,\ell, \quad \ell = 1,\dots,n-d, \end{aligned}$$

as shown in Figure 3, and it follows by induction that $P_i^{n-d} = 1$ for $i = 0,\dots,n$. Therefore,

$$f(x) = (-1)^{n-d}\pi(x)\sum_{i=0}^{n-d}\lambda_i(x)p_i(x), \qquad g(x) = (-1)^{n-d}\pi(x)\sum_{i=0}^{n-d}\lambda_i(x),$$

and a final division of these two components of $Q_0^n(x)$ gives

$$r(x) = \frac{f(x)}{g(x)}.$$

Another option is to evaluate the interpolant r by modifying all the rows of Neville's algorithm, as shown in Figure 4, and to use the initial data

$$R_i^0(x) = w_i(f_i, 1) = (w_i f_i, w_i), \qquad i = 0,\dots,n.$$

We then compute the rows of the pyramid from bottom to top as

$$R_i^\ell(x) = \eta_i^{n-\ell+1}(x_{i+\ell} - x)R_i^{\ell-1}(x) + \eta_{i+1}^{n-\ell+1}(x - x_i)R_{i+1}^{\ell-1}(x), \qquad i = 0,\dots,n-\ell,$$

for $\ell = 1,\dots,n$ with η_i^ℓ defined as above, resulting in the value

$$R_0^n(x) = \big(\tilde{f}(x), \tilde{g}(x)\big).$$

With arguments similar to the ones above, it can be shown that

$$\tilde{f}(x) = (-1)^n\pi(x)\sum_{i=0}^{n}\frac{(-1)^i}{x-x_i}w_i f_i, \qquad \tilde{g}(x) = (-1)^n\pi(x)\sum_{i=0}^{n}\frac{(-1)^i}{x-x_i}w_i,$$

and a final division gives

$$r(x) = \frac{\tilde{f}(x)}{\tilde{g}(x)}.$$

[2]For the sake of simplicity, we tacitly follow the convention that $P_i^\ell = 0$ for $i < 0$ and $i > \ell$.

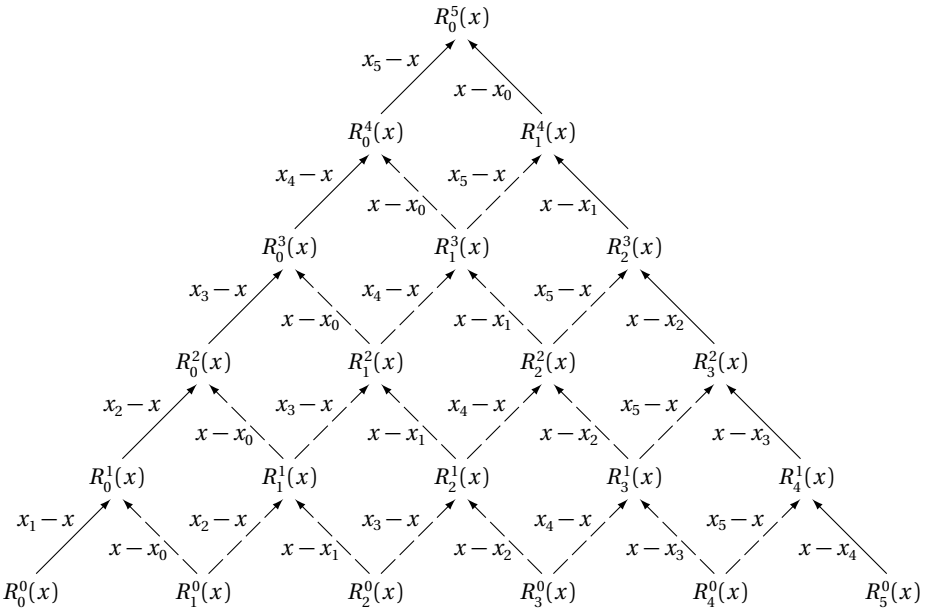

Figure 4: Example of the pyramid algorithm for general barycentric rational interpolation with $n=5$.

Note that this second algorithm works for any set of weights w_i, hence for any rational interpolant of degree (n,n).

Clearly, both algorithms require $O(n^2)$ steps and are thus slower and possibly not as robust as the $O(n)$ evaluation of $r(x)$ with the barycentric form (2), but we believe that this formulation sheds some interesting new light on the idea of barycentric rational interpolation. We would further like to point out that the algorithm in Figure 2 bears a strong resemblance to the algorithm of Barry and Goldman [1] for evaluating Catmull–Rom splines. Both algorithms first carry out a few rounds of Neville's algorithm to compute the local polynomial interpolants $p_i(x)$, but they differ in the way these values are combined while going through the top rows of the pyramid: for Catmull–Rom splines we blend the local polynomials with B-spline basis functions, and for Floater–Hormann interpolants we use $\lambda_i/\sum_{j=0}^{n-d}\lambda_j$ as blending functions.

3 Computing the weights

Comparing the two algorithms in Figures 2 and 4, we notice that they are identical in the top $n-d$ rows and that the input data $Q_i^0(x)$ and $R_i^0(x)$ differ by the factors w_i. This observation suggests that there might exist an efficient pyramid algorithm for computing the Floater–Hormann weights. Indeed, we can determine the weights w_i if we start with the values

$$V_i^d = 1, \qquad i = 0,\ldots,n-d$$

and then work our way down the bottom d rows of the pyramid, using the normalization factors from Neville's algorithm, to compute[3]

$$V_i^\ell = \frac{V_{i-1}^{\ell+1}}{x_{i+\ell}-x_{i-1}} + \frac{V_i^{\ell+1}}{x_{i+\ell+1}-x_i}, \qquad i = 0,\ldots,n-\ell, \tag{4}$$

for $\ell = d-1, d-2,\ldots,0$, as shown in Figure 5. To see that $V_i^0 = w_i$ for $i=0,\ldots,n$, consider the values

$$U_i^\ell = \sum_{j=\max(0,i-\ell)}^{\min(i,n-\ell)} V_j^\ell \prod_{k=j,k\neq i}^{j+\ell} \frac{1}{|x_i-x_k|}, \qquad i=0,\ldots,n, \quad \ell=0,\ldots,d. \tag{5}$$

For these values we clearly have $U_i^0 = V_i^0$ and $U_i^d = w_i$ for $i=0,\ldots,n$, and we further show that $U_i^\ell = U_i^{\ell+1}$ for any $\ell = 0,\ldots,d-1$. The main idea is first to expand each addend A_j^ℓ of U_i^ℓ using (4) and then to augment

[3] For the sake of simplicity, we tacitly follow the convention that $V_i^\ell = 0$ for $i<0$ and $i>n-\ell$.

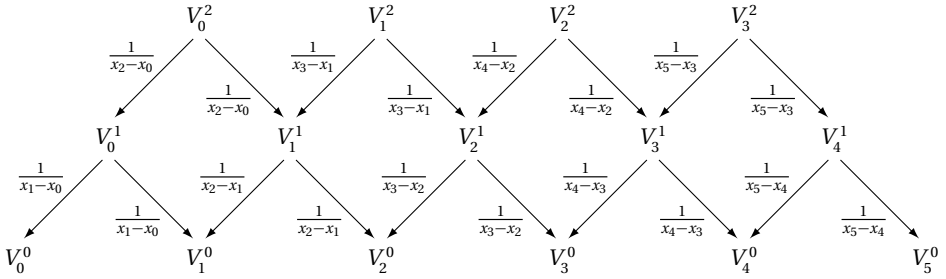

Figure 5: Example of the pyramid algorithm for computing the Floater–Hormann weights w_i in (3) with $n=5$ and $d=2$.

the products by one factor,

$$A_j^\ell = V_j^\ell \prod_{k=j,k\neq i}^{j+\ell} \frac{1}{|x_i-x_k|} = \left(\frac{V_{j-1}^{\ell+1}}{x_{j+\ell}-x_{j-1}} + \frac{V_j^{\ell+1}}{x_{j+\ell+1}-x_j}\right) \prod_{k=j,k\neq i}^{j+\ell} \frac{1}{|x_i-x_k|}$$
$$= \underbrace{V_{j-1}^{\ell+1} \frac{x_i-x_{j-1}}{x_{j+\ell}-x_{j-1}} \prod_{k=j-1,k\neq i}^{j+\ell} \frac{1}{|x_i-x_k|}}_{=B_j^\ell} + \underbrace{V_j^{\ell+1} \frac{x_{j+\ell+1}-x_i}{x_{j+\ell+1}-x_j} \prod_{k=j,k\neq i}^{j+\ell+1} \frac{1}{|x_i-x_k|}}_{=C_j^\ell}.$$

Next observe that the terms C_j^ℓ from A_j^ℓ and B_{j+1}^ℓ from A_{j+1}^ℓ sum up to the addend $A_j^{\ell+1}$ of $U_i^{\ell+1}$,

$$C_j^\ell + B_{j+1}^\ell = V_j^{\ell+1} \frac{x_{j+\ell+1}-x_i}{x_{j+\ell+1}-x_j} \prod_{k=j,k\neq i}^{j+\ell+1} \frac{1}{|x_i-x_k|} + V_j^{\ell+1} \frac{x_i-x_j}{x_{j+\ell+1}-x_j} \prod_{k=j,k\neq i}^{j+\ell+1} \frac{1}{|x_i-x_k|}$$
$$= V_j^{\ell+1} \prod_{k=j,k\neq i}^{j+\ell+1} \frac{1}{|x_i-x_k|} = A_j^{\ell+1}.$$

Finally, notice that $B_0^\ell = 0$, because $V_{-1}^{\ell+1} = 0$, and $B_{i-\ell}^\ell = A_{i-\ell-1}^{\ell+1}$ for $i > \ell$, and similarly $C_{n-\ell}^\ell = 0$, because $V_{n-\ell}^{\ell+1} = 0$, and $C_i^\ell = A_i^{\ell+1}$ for $i < n-\ell$. If we now denote the lower and upper bounds of the summation index j in (5) by

$$\alpha_i^\ell = \max(0, i-\ell), \qquad \beta_i^\ell = \min(i, n-\ell)$$

and distinguish the different cases of the index i, where either $\alpha_i^\ell = 0$ or $\alpha_i^\ell = i-\ell$ and either $\beta_i^\ell = i$ or $\beta_i^\ell = n-\ell$, we find that in all cases

$$U_i^\ell = \sum_{j=\alpha_i^\ell}^{\beta_i^\ell} A_j^\ell = B_{\alpha_i^\ell}^\ell + \sum_{j=\alpha_i^\ell+1}^{\beta_i^\ell} B_j^\ell + \sum_{j=\alpha_i^\ell}^{\beta_i^\ell-1} C_j^\ell + C_{\beta_i^\ell}^\ell = B_{\alpha_i^\ell}^\ell + \sum_{j=\alpha_i^\ell}^{\beta_i^\ell-1} A_j^{\ell+1} + C_{\beta_i^\ell}^\ell = \sum_{j=\alpha_i^{\ell+1}}^{\beta_i^{\ell+1}} A_j^{\ell+1} = U_i^{\ell+1}.$$

Therefore,

$$V_i^0 = U_i^0 = U_i^1 = \cdots = U_i^d = w_i,$$

which asserts that the Floater–Hormann weights are determined by the algorithm above in $O(nd)$ steps.

References

[1] P. J. Barry and R. N. Goldman. A recursive evaluation algorithm for a class of Catmull–Rom splines. *ACM SIGGRAPH Computer Graphics*, 22(4):199–204, Aug. 1988.

[2] S. Bochkanov. ALGLIB 3.10.0 User Guide – Interpolation and fitting – Rational interpolation. `http://www.alglib.net/interpolation/rational.php`, Aug. 2015. [Online; accessed 14-December-2015].

[3] M. S. Floater and K. Hormann. Barycentric rational interpolation with no poles and high rates of approximation. *Numerische Mathematik*, 107(2):315–331, Aug. 2007.

[4] R. Goldman. *Pyramid Algorithms: A Dynamic Programming Approach to Curves and Surfaces for Geometric Modeling*. The Morgan Kaufmann Series in Computer Graphics and Geometric Modeling. Morgan Kaufmann, San Francisco, 2003.

[5] S. Teukolsky, B. P. Flannery, W. T. Vetterling, and W. H. Press. *Numerical Recipes in C: The Art of Scientific Computing*, chapter 3.4.1, pages 127–129. Cambridge University Press, New York, third edition, 2007.