This is a function F in $2M-1$ variables $b_1, \ldots, b_{M-1}, a_1, \ldots, a_M$ and their extrema are reached at values with $\nabla F = \vec{0}$. By the fundamental theorem of calculus, we have

$$\frac{\partial F}{\partial b_j} = (a_j - b_j)^2 g(b_j) - (a_{j+1} - b_j)^2 g(b_j) \qquad \text{for } j = 1, 2, \ldots, M-1, \tag{2.22}$$

and differentiating under the integral

$$\frac{\partial F}{\partial a_j} = 2 \int_{I_j} (a_j - x)^2 g(x)\, dx \qquad \text{for } j = 1, 2, \ldots, M. \tag{2.23}$$

If $g(b_j) \neq 0$, the vanishing of (2.22) is equivalent to $2b_j = a_j + a_{j+1}$. The *Lloyd-Max algorithm* [Llo82] [Max60] takes this information to solve $\nabla F = \vec{0}$ by successive approximations. Namely, the algorithm starts with any educated guess for the representation points and applies successively the following formulas (in the indicated order)

(2.24)

$$b_j = \frac{a_j + a_{j+1}}{2} \quad \text{for } j = 1, 2, \ldots, M-1 \quad \text{and} \quad a_j = \frac{\int_{I_j} x g(x)\, dx}{\int_{I_j} x g(x)\, dx} \quad \text{for } j = 1, 2, \ldots, M.$$

Note that the second set of equations solves (2.23) equated to 0. The algorithm ends when the values of the b_j's and a_j's stabilize with certain precision. A drawback of the algorithm is that it could converge to a critical point different from one in which the absolute minimum of F is attained. One can cook some counterexamples of this kind [Gal06] choosing a distribution function g with some valleys but anyway one trusts the educated initial guess should avoid this problem.

Suggested Readings. As you see, from the mathematical point of view, quantization is not a big deal. Probably you do not need extra bibliography. I only dare to repeat that in [You14] you can find the odd terminology used by engineers.

2.1.3 Data approximation

In practice, Theorem 2.1.1 allows to reconstruct a band limited function from their sampled values but as we mentioned in connection with Papoulis-Gerchberg algorithm and (2.8), its practical efficiency is objectionable in several situations. On the other hand, after quantization and digital processing very rarely it is judicious to assume that the resulting data are exact values of a band limited function.

Here we approach D/A conversion in a simple general mathematical form: We have discrete (digital) data and we want to approximate them with a somewhat smooth function. We restrict ourselves to 1D samples, as before, then have in mind sound signals rather than images.

You know how to do it, finite samples, simple functions like polynomials... connect the dots, literally. It is interpolation from basic courses of numerical analysis. You have some *interpolation nodes* $x_0 < x_1 < \cdots < x_n$ and you want to find a polynomial P such that $P(x_j) = y_j$ for some given y_j. There is a nice theorem, with a short beautiful proof, providing the error with respect to the purported smooth function connecting the dots.

Theorem 2.1.3. *Given $x_0 < x_1 < \cdots < x_n$ and $f \in C^{n+1}([x_0, x_n])$ there exists a unique polynomial P of degree at most n such that $P(x_j) = f(x_j)$. For each $x \in [x_0, x_n]$ there exists $\xi \in (x_0, x_n)$ such that*

$$f(x) - P(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!} \prod_{j=0}^{n} (x - x_j). \tag{2.25}$$

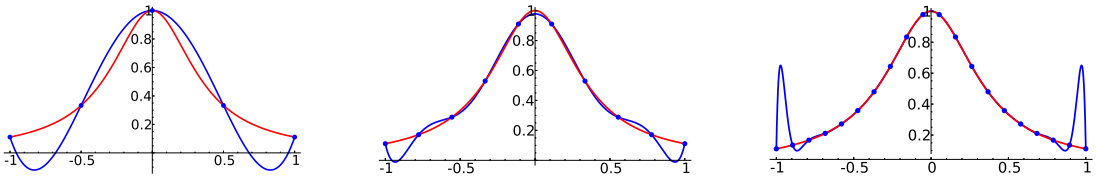
Proof. The existence of P follows taking $P = \sum f(x_j) L_k$ where L_k is the *Lagrange polynomial* $L_k(x) = \prod_{j \neq k} (x - x_k)/(x_j - x_k)$ that verifies $L_k(x_j) = 0$ for $j \neq k$ and $L_k(x_k) = 1$. The uniqueness follows because a polynomial of degree at most n with $n + 1$ zeros is identically zero.

For $x = x_j$, (2.25) is trivial. In the rest of the cases, for $x \in [x_0, x_n]$ fixed different from the interpolation points, define $g : [x_0, x_n] \longrightarrow \mathbb{R}$ by

$$g(t) = f(t) - P(t) - (f(x) - P(x)) \prod_{j=0}^{n} \frac{t - x_j}{x - x_j}. \tag{2.26}$$

It satisfies $g(x_0) = \cdots = g(x_n) = g(x) = 0$ and applying $n + 1$ times Rolle's theorem we conclude $g^{(n+1)}(\xi) = 0$ (isn't it beautiful?) and it gives the result because $P^{(n+1)} = 0$ and $(n + 1)!$ is the $(n + 1)$-derivative of $\prod(t - x_j)$. □

The $(n + 1)!$ grows to the speed of light (even faster because somebody said that the latter is constant). Then probably you expect that for "smooth" data the more (degree) the merrier. Let us check what happens with $f(x) = (1 + 8x^2)^{-1}$ in $[-1, 1]$ when we take N equally spaced nodes with $N = 5$, 10 and 15. Behold the results!



For $N = 50$ the error has peaks of order 10^5. You are seeing the so-called *Runge's phenomenon* [KC96]. the rough idea under this weird situation is that for an analytic function with bounded convergence radius the Taylor coefficients behave like a power (by the root test) then you should not expect too much from the fraction in (2.25) and especially near the extremes the product involves a factorial in the numerator.

Definitively, it is not a good idea to use large degree polynomials to approximate functions even if they are C^∞. What about using low degree polynomials?

A first approach is joining the dots as kids, using straight lines (the same method to define the length of rectifiable curve, if you prefer a more scientific precedent). Not bad but we get corners. It is up to us to decide what kind of regularity we want but some people would say that in graphical representations one can see at first glance first derivatives (growth, monotonicity) and second derivatives (curvature). Let us focus then

on $C^2([x_0, x_n])$ interpolation. With something piecewise linear we get a continuous result, with quadratic we get C^1 and we need piecewise cubic to reach C^2. Given the interpolation nodes $x_0 < x_1 < \cdots < x_n$ and their images y_j, we define the space

$$C_n^2 = \{f \in C^2([x_0, x_n]) \,:\, f(x_j) = y_j,\ 0 \le j \le n \text{ and } \ f''(x_0) = f''(x_n) = 0\}. \tag{2.27}$$

The functions in this space that coincide with a polynomial of degree at least 3 in each interval $[x_j, x_{j+1}]$ are called *cubic splines*, the adjective *natural* is added to indicate the boundary condition $f''(x_0) = f''(x_n) = 0$. There are other possibilities not discussed here.

How do we know that we can find a (natural) cubic spline for any interpolation nodes and corresponding values? It is a theorem but a simple count suggests that it is true: Each polynomial of degree at most three has 4 coefficients and we have one of them for each interval $[x_j, x_{j+1}]$ then we have in total $4n$ unknowns. On the other hand, in the inner nodes $s \in C^2$ imposes $s(x_j^-) = s(x_j^+) = y_j$, $s'(x_j^-) = s'(x_j^+)$ and $s''(x_j^-) = s''(x_j^+)$ that make $4(n-1)$ linear equations. At the boundary nodes $s(x_0) = y_0$, $s(x_n) = y_n$, $s''(x_0) = s''(x_n) = 0$ give 4 more linear equations. A linear system with as many equations as unknowns is very likely to have a unique solution. This is the case for cubic splines and indeed the resulting linear system suits special known algorithms of numerical linear algebra that allow to deal with a large number of nodes. In the following proof we exemplify the situation in the case of equally spaced nodes. With little changes, it extends to the general case [SB02, §2.4.2].

Theorem 2.1.4. *Given $x_0 < x_1 < \cdots < x_n$ and $\{y_j\}_{j=0}^n$, there exists a unique cubic spline $s \in C_n^2$.*

Proof (for equally spaced nodes). After a linear change it suffices to consider $x_j = j$. Let us say that the restriction of s to the interval $[j, j+1]$ is $S_j(t+j)$ where $S_j : [0,1] \longrightarrow \mathbb{R}$ is the polynomial

$$S_j(t) = s_1^j t^3 + s_2^j t^2 + s_3^j t + s_4^j \qquad \text{with} \quad 0 \le j < n. \tag{2.28}$$

The upper indexes, of course, do not indicate powers here.

Let us write the conditions mentioned above. Firstly, s must join the interpolation points (x_j, y_j) then for $0 \le j < n$

$$s_4^j = y_j, \tag{2.29}$$

$$s_1^j + s_2^j + s_3^j + s_4^j = y_{j+1}. \tag{2.30}$$

On the other hand, $s \in C^2$ imposes $S_j'(0) = S_{j-1}'(1)$ and $S_j''(0) = S_{j-1}''(1)$. Equivalently, for $0 < j < n$

$$s_3^j = 3s_1^{j-1} + 2s_2^{j-1} + s_3^{j-1}, \tag{2.31}$$

$$2s_2^j = 6s_1^{j-1} + 2s_2^{j-1}. \tag{2.32}$$

Finally, the boundary values of the second derivative give

$$2s_2^0 = 0, \tag{2.33}$$

$$6s_1^{n-1} + 2s_2^{n-1} = 0. \tag{2.34}$$

In principle the linear system (2.29)–(2.34) seems messy but it becomes quite simple if we employ as unknowns the values $D_j = s'(j)$ that coincide with both sides of (2.31). Subtracting (2.30) and (2.29), it follows $s_1^j + s_2^j + s_3^j = y_{j+1} - y_j$ and on the other hand $D_{j+1} + D_j = (3s_1^j + 2s_2^j + s_3^j) + s_3^j$. Therefore $s_1^j = D_{j+1} + D_j - 2(y_{j+1} - y_j)$. In the same way we can get s_2^j. Summing up, if we find D_j we shall have proved that the linear system (2.29)–(2.34) has the solution

$$\begin{cases} s_1^j = D_{j+1} + D_j - 2(y_{j+1} - y_j), & s_3^j = D_j \\ s_2^j = -D_{j+1} - 2D_j + 3(y_{j+1} - y_j), & s_4^j = y_j. \end{cases} \tag{2.35}$$

To get an equation for D_j, we employ (2.32). Substituting (2.35) we obtain

$$D_{j-1} + 4D_j + D_{j+1} = 3(y_{j+1} - y_{j-1}), \tag{2.36}$$

for $0 < j < n$. For $j = 0$ and $j = n$ it has to be modified according (2.33) and (2.34) to

$$2D_0 + D_1 = 3(y_1 - y_0) \qquad \text{and} \qquad D_{n-1} + 2D_n = 3(y_n - y_{n-1}). \tag{2.37}$$

The linear system (2.36)–(2.37) has a *tridiagonal matrix*, one having nonzero elements only on the main diagonal and on the neighboring diagonals above and below. It is easy to see that it is nonsingular (for instance applying Gaussian elimination or computing the determinant with an inductive argument [Mui60]). There are very efficient methods to solve linear systems with this kind of matrices [SB02]. Once we know that D_j are uniquely determined, the same apply for the coefficients s_k^j thanks to (2.35). □

We now have a theorem that states that cubic splines exist and digging in the proof they are easy to compute numerically. Are they useful? Have we avoided with them the large wobbles of Runge's phenomenon? The answer is absolutely yes. It turns out that cubic splines are the "less curved" function connecting the interpolation points. In mathematical terms

Theorem 2.1.5. *Let s be the unique spline in C_n^2. Then for any other function $f \in C_n^2$ we have $\|s''\|_2 \leq \|f''\|_2$ with equality only if $s = f$.*

Proof. We have

$$\|f''\|_2^2 = \|f'' - s''\|_2^2 + \|s''\|_2^2 + 2\int_{x_0}^{x_n} s''(f'' - s''). \tag{2.38}$$

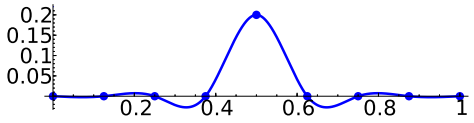
If the integral is zero, we get the first part of the result. Let us check this point. On each interval $[x_j, x_{j+1}]$ the spline s is a polynomial s_j, integrating by parts on each of these intervals

$$\int_{x_0}^{x_n} s''(f'' - s'') = \sum_{j=0}^{n-1} \left(f'(x_{j+1})s''(x_{j+1}) - f'(x_j)s''(x_j)\right) - \sum_{j=0}^{n-1} \int_{x_j}^{x_{j+1}} s_j'''(f' - s'). \tag{2.39}$$

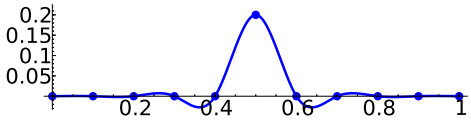
The first sum telescopes to 0. The last integral is also zero because s_j''' is constant and $f - s$ vanishes at x_j and x_{j+1}.

If $\|s''\|_2 = \|f''\|_2$ then (2.38) gives $\|f'' - s''\|_2 = 0$ and f and s may differ in a quadratic function but $f''(x_0) = s''(x_0) = 0$, $f(x_0) = s(x_0)$, $f(x_1) = s(x_1)$ imply that they actually coincide. □

Let us try a couple of numerical examples. Consider $2N+1$ nodes in $[0,1]$ and impose $s(x_j) = 0$ except for the central point $s(x_N) = 0.2$. For $N = 4$ and $N = 5$ (9 and 11 nodes), we obtain



Nine nodes



Eleven nodes

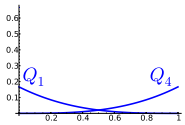
In both cases we see that the first and the last pieces are almost flat. For $N = 4$ we have $s'(0) \approx -6 \cdot 10^{-3}$ and for $N = 5$ we have $s'(0) \approx 1.6 \cdot 10^{-3}$.

After this little long distance influence, one may wonder whether there exists a nontrivial compactly supported cubic spline b, meaning that it is identically zero outside an interval determined by two nodes $x_{j_1} < x_{j_2}$, hence it must hold $b''(x_{j_k}) = b'(x_{j_k}) = b(x_{j_k}) = 0$. As before, we restrict ourselves to the case of equally spaced nodes. Some calculations show that there is no solution for $j_2 - j_1 < 4$. On the other hand, for $x_j = j - 2$, $0 \le j \le n = 4$ we have

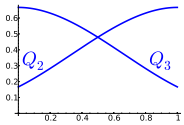
(2.40)

$$b(t) = Q_j(t - x_j) \quad \text{if } x_j \le t \le x_{j+1} \text{ with } \begin{cases} Q_1(t) = \frac{1}{6}t^3, & Q_3(t) = Q_2(1-t), \\ Q_2(t) = \frac{1}{6}(t+1)^3 - \frac{2}{3}t^3, & Q_4(t) = Q_1(1-t), \end{cases}$$

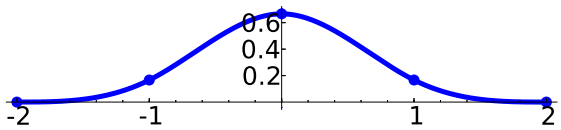
that vanishes to order 3 at the end nodes and can be safely extended as zero outside its support. The graphics of the pieces Q_j composing this spline and of the spline itself are



Q_1 and Q_4



Q_2 and Q_3



The spline b

This cubic spline is, except for scaling, the only one supported in three equally spaced consecutive nodes. The splines like this, having minimal support for a given regularity are called *B-splines*. What is the big deal about them? It is related to applications. Let us say that we have to interpolate f at a huge number of consecutive integer nodes $x_0 < x_1 < \cdots < x_n$. It requires solving the linear system indicated in the proof of Theorem 2.1.4. Moving a little a value y_j, as we have seen, has little influence except for few nearby nodes, then one may consider a cheap alternative that does not require any numerical linear algebra

$$s_f(x) = \sum_j f(x_j) b(t - x_j) \tag{2.41}$$

where we use integer translated copies of b as a basis (the "B" in "B-splines" stands for "basis"). As b only overlaps with four copies, we have the formula

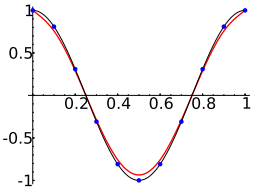
$$s_f(x) = \sum_{k=1}^{4} Q_k(t - x_j) f(x_{j+3-k}) \qquad \text{for} \quad x_j \le t \le x_{j+1} \quad \text{with} \quad 0 < j < n-1. \tag{2.42}$$

We have to take a decision about what happens on the boundary $j = 0$ and $j = n - 1$. A natural one is to keep (2.42) in these cases introducing two artificial nodes x_{-1} and x_{n+1} at which we assume f takes the linear extrapolated values i.e., $f(x_{-1}) = 2f(x_0) - f(x_1)$ and $f(x_{n+1}) = 2f(x_n) - f(x_{n-1})$.

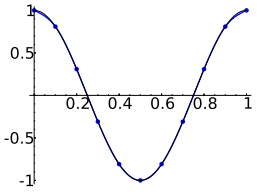
Note that each evaluation requires only 4 multiplications by the pieces of the B-spline, instead of using $n + 1$ coefficients coming from a previously solved linear system. The drawback is that (2.42) does not perform actual interpolation. At the nodes we have

$$s_f(x_j) = \sum_{k=1}^{4} Q_k(0) f(x_{j+3-k}) = \frac{f(x_{j-1}) + 4f(x_j) + f(x_{j+1})}{6}. \tag{2.43}$$

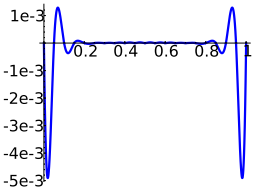
What is the point of having a hyper-speed non-interpolating formula like (2.42)? A possible answer is that after quantization we have already introduced errors everywhere and imposing error free interpolation of error affected data is in some occasions to use a sledgehammer to crack a nut. Moreover for smooth data, B-splines give a good approximation to spline exact approximation. For instance, if we interpolate $f(x) = \cos(2\pi x)$ in $[0, 1]$ with $n = 10$ the maximal error is like $2 \cdot 10^{-2}$ and with B-splines $6 \cdot 10^{-2}$. Increasing n to 20 the figures are $5 \cdot 10^{-3}$ and $1.6 \cdot 10^{-2}$ and for $n = 40$, 10^{-3} and $4 \cdot 10^{-3}$. It is fair to mention that the error in the cubic spline interpolation accumulates near the endpoints and in the inner point the approximation is by far better. This is because $f''(0), f''(1) \neq 0$ while natural splines satisfy $s''(0) = s''(1) = 0$. Anyway, the error using B-splines is small taking into account the simplicity of the method.



n = 10, B-splines



n = 10, interpolation



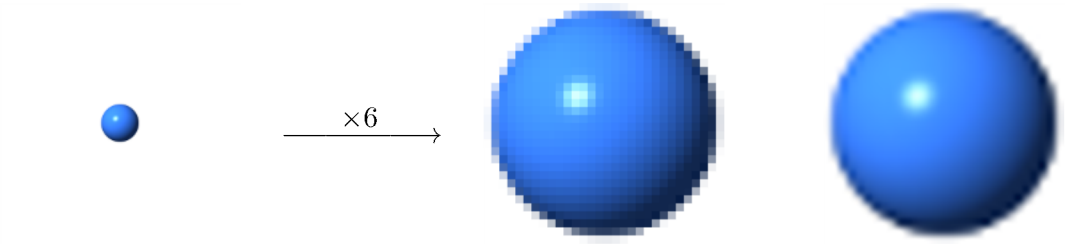
n = 20, interpolation error

To emphasize that this is practical, let us consider an example related to GIMP (the free, cross-platform and open-source competitor of Adobe Photoshop). If you try to scale an image with it, under the name "Quality" there is a little menu to choose None, Linear, Cubic and Sinc. By default it is selected Cubic that, according to the documentation, "produces the best results". Why on earth are there several possibilities for this simple operation? In principle to enlarge a photo making the sides six times longer is just passing the color of the point (x, y) to the point $(6x, 6y)$. Yes, it is but $f(\vec{x}) = 6\vec{x}$ is a bijection when applied on $\mathbb{R}^2$ and not when applied on $\mathbb{Z}^2$ and the pixels are labeled by integers. The result would be an image plenty of holes. OK, let us take the units six times bigger, in this way we

apply the pixel (m, n) into the square of pixels $(6m + k, 6n + l)$ with $k, l \in \{0, 1, \ldots, 5\}$. This corresponds to None in GIMP. It seems the only natural solution but the results are in general visually poor because we see the enlarged edges of the squares of the pixels. A better solution is to assign the color to the pixels $(6m, 6n)$, use them as nodes and interpolate the rest of the pixels. Exact cubic interpolation would be expensive (an image 640×400 requires 256000 nodes) while B-spline approximation is affordable. How can we manage B-splines in this $2D$ setting? If $f = f(x, y)$ is the function giving the color, (2.42) generalizes promptly to approximate it by

(2.44)
$$\sum_{k=1}^{4}\sum_{l=1}^{4} Q_k(x - x_j)Q_l(y - y_m)f(x_{j+3-k}, y_{m+3-l}) \qquad \text{for} \quad x_j \le t \le x_{j+1}, \quad y_m \le t \le y_{m+1}.$$

This is the Cubic method. As you suspect, Linear means piecewise linear approximation. Finally Sinc is an interpolation method related to the sinc function, a kind of smoothing of (2.8). Here you can see an example of enlarging an image without and with B-splines. Which one do you prefer?



Original 30×30 None 180×180 Cubic 180×180

For the avid reader, a last brief comment about non-interpolating cubic curves. Probably you have heard the name *Bézier curves*. They are based in the following fact: Given $P_0, P_1, P_2, P_3 \in \mathbb{R}^2$, the curve $\sigma : [0, 1] \longrightarrow \mathbb{R}$

(2.45)
$$\sigma(t) = (1 - t)^3 P_0 + 3t(1 - t)^2 P_1 + 3t^2(1 - t)P_2 + t^3 P_3.$$

joins P_0 and P_1 and pass somewhat close to P_1 and P_2. The segments P_0P_1 and P_2P_3 are tangent to the curve. The Bézier curves are composed by curves like this. Many interactive applications use them because it is quite intuitive to use P_1 and P_2 as *control points* that when moved change locally the aspect of the curve.

Suggested Readings. Lagrange interpolation, splines and B-splines are discussed in the second chapter of [SB02]. Look up this book for a mathematically spotless treatment of several numerical analysis methods. There is a lot of information about Bézier curves and their relatives on the internet and on texts oriented to the applications. I have found very clear the exposition in [Bus03].

2.1.4 Dithering

If I say that *dithering* consists of improving A/D conversion adding noise to the analog signal you will think I am crazy. We are not talking about me but truly this seems idiotic