Interpolation

Atkinson Chapter 3, Stoer & Bulirsch Chapter 2, Dahlquist & Bjork Chapter 4

Topics marked with * are not on the exam

1 Polynomial interpolation, introduction. Let $\{x_i\}_0^n$ be distinct real numbers and let $\{y_i\}_0^n$ be real. The interpolation problem attempts to find a function $p(x)$ with the property $p(x_i) = y_i$ for all i. Clearly there are many solutions. For example $x_0 = -1$, $x_1 = 1$, $y_0 = y_1 = 1$ could be interpolated by $p(x) = 1$ or by $p(x) = x^2$. We will consider solving the interpolation problem (mainly in 1D) where we restrict p to be in one of a few finite-dimensional function spaces. To begin, we consider p polynomial. We have $n+1$ conditions, and a polynomial of degree m has $m+1$ coefficients so try to fit an n^{th} degree polynomial

$$\begin{aligned} p(x_0) &= a_0 + a_1 x_0 + \ldots + a_n x_0^n = y_0 \\ p(x_1) &= a_0 + a_1 x_1 + \ldots + a_n x_1^n = y_1 \\ p(x_2) &= a_0 + a_1 x_2 + \ldots + a_n x_2^n = y_2 \\ &\vdots \\ p(x_n) &= a_0 + a_1 x_n + \ldots + a_n x_n^n = y_n \end{aligned}$$

This is a linear system for the unknown coefficients a_i with RHS $\{y_i\}$ and coefficient matrix

$$(\mathbf{V})_{i,j} = x_{i-1}^{j-1}, \quad i,j = 1,\ldots,n+1.$$

This is called a Vandermonde matrix (sometimes people say that $\mathbf{V}^T$ is the Vandermonde matrix). Iff it's invertible then the polynomial interpolation problem has a unique solution for polynomials of degree $\leq n$.

- If it's invertible then there is a unique set of coefficients a_i that define a polynomial of degree $\leq n$ that solves the interpolation problem.
- If there's a unique polynomial of degree $\leq n$ that solves the interpolation problem, then it must be the only solution to the equations above.

An invertible matrix must have linearly independent rows, which shows why it is important to have *distinct* x_i: **If two nodes are equal $x_i = x_j$, $i \neq j$ then the matrix is not invertible.** To prove that the matrix is invertible whenever the nodes are distinct we will not deal directly with the Vandermonde matrix; instead construct a solution.

2 Lagrange. Consider

$$\ell_i(x) = \frac{(x-x_0)(x-x_1)\cdots(x-x_{i-1})(x-x_{i+1})\cdots(x-x_n)}{(x_i-x_0)(x_i-x_1)\cdots(x_i-x_{i-1})(x_i-x_{i+1})\cdots(x_i-x_n)}.$$

In order to be well-defined we need $x_i \neq x_j$ for $i \neq j$, or there would be a 0 in the denominator.

By construction, this is a polynomial of degree n with $\ell_i(x_j) = \delta_{ij}$. Now consider (Lagrange's formula)

$$p(x) = \sum_{i=0}^{n} y_i \ell_i(x).$$

This is also a polynomial of degree $\leq n$, with the property $p(x_i) = y_i$.

We have shown by construction that the interpolation problem *has* a solution for any set of data $\{y_i\}$. The fact that the square Vandermonde system has a solution for any data means that the Vandermonde system is nonsingular/intertible. The solution to an invertible linear system is unique, so it the solution to the polynomial interpolation problem must also be unique.

3 Let's step back and consider our two methods so far. First, we are seeking $p(x)$ in a finite-dimensional vector space: polynomials of degree $\leq n$. Any vector space has an infinite number of bases; in the Vandermonde approach we used the standard 'monomial' basis: $\{1, x, \ldots, x^n\}$. The solution of the linear Vandermonde system is the coefficients of the polynomial in the standard basis.

The Lagrange method uses a different basis: $\{\ell_0(x), \ldots, \ell_n(x)\}$. We should prove that this is a basis. Linear independence is easy since $\ell_i(x_j) = \delta_{ij}$. How do we know that these functions span the vector space of polynomials of degree $\leq n$ (whenever the nodes are distinct)? We want to know whether any polynomial $q(x)$ of degree $\leq n$ can be written as a linear combination of the Lagrange polynomials. The answer is subtle but obvious in retrospect:

$$q(x) = \sum_i q(x_i)\ell_i(x).$$

The fact that we have equality between the left and right hand sides of this equation is guaranteed by the fact that the interpolation problem has a unique solution: the RHS is the *unique* interpolating polynomial, but the LHS is also an interpolating polynomial.

Remark: The interpolating polynomial is unique. You can write it as a sum in any basis you want, but it is still the same polynomial.

It's a lot easier to solve the polynomial interpolation problem using the Lagrange basis: In the monomial basis you have to solve a dense linear system for the expansion coefficients; in the Lagrange basis the expansion coefficients are just y_i. But you generally want to *use* the solution once you have it, i.e. you want to evaluate $p(x)$ at points other than x_i. To evaluate using the Lagrange basis you need to evaluate $n+1$ polynomials $\ell_i(x)$ and then sum them up, whereas to evaluate using the standard basis you only need to evaluate one polynomial. So: monomial basis is hard to find but easier to use, and the Lagrange basis is easier to solve but harder to use.

Is there another basis that is both easy to solve and easy to use? Suppose the functions $\{\phi_i(x)\}_{i=1}^n$ are a basis for polynomials of degree $\leq n$. The interpolating polynomial is

$$p_n(x) = \sum_{i=0}^{n} c_i\phi_i(x)$$

and the system of equations for the expansion coefficients c_i is

$$\begin{aligned}
p(x_0) &= c_0\phi_0(x_0) + c_1\phi_1(x_0) + \ldots + c_n\phi_n(x_0) = y_0 \\
p(x_1) &= c_0\phi_0(x_1) + c_1\phi_1(x_1) + \ldots + c_n\phi_n(x_1) = y_1 \\
&\vdots \\
p(x_n) &= c_0\phi_0(x_n) + c_1\phi_1(x_n) + \ldots + c_n\phi_n(x_n) = y_n
\end{aligned}$$

Suppose that the basis functions $\phi_i(x)$ are related to the monomials as follows

$$\begin{pmatrix} \phi_0(x) \\ \vdots \\ \phi_n(x) \end{pmatrix} = \mathbf{S}^T \begin{pmatrix} 1 \\ \vdots \\ x^n \end{pmatrix}$$

where the matrix $\mathbf{S}$ is invertible (it must be, if the ϕ_i are a basis). The coefficients of $p(x)$ in the two bases are related by

$$p(x) = \boldsymbol{a}^T \begin{pmatrix} 1 \\ \vdots \\ x^n \end{pmatrix} = \boldsymbol{c}^T \begin{pmatrix} \phi_0(x) \\ \vdots \\ \phi_n(x) \end{pmatrix} = \boldsymbol{c}^T\mathbf{S}^T \begin{pmatrix} 1 \\ \vdots \\ x^n \end{pmatrix}$$

$$(\boldsymbol{a}^T - \boldsymbol{c}^T\mathbf{S}^T) \begin{pmatrix} 1 \\ \vdots \\ x^n \end{pmatrix} = 0$$

i.e.

$$\mathbf{S}\boldsymbol{c} = \boldsymbol{a}.$$

Coming back to the original Vandermonde system we see that

$$\mathbf{V}\boldsymbol{a} = \boldsymbol{y} \Rightarrow \mathbf{VS}\boldsymbol{c} = \boldsymbol{y}.$$

This shows that using a different basis is equivalent to a right-preconditioner on the Vandermonde system. For example, if we use the Lagrange polynomials as our basis, then we know that $c_i = y_i$:

$$\phi_i(x) = \ell_i(x), \;\; p(x) = \sum_i y_i \ell_i(x) = \sum_i c_i \phi_i(x)$$

so

$$\mathbf{VS} = \mathbf{I}, \;\; \mathbf{S} = \mathbf{V}^{-1}.$$

4 Return to our question: Is there another basis besides monomials and the Lagrange basis that is both easy to solve and easy to use? Yes, there are many options here. The classic example is the 'Newton basis.' Newton uses a basis of the form

$$\phi_0(x) = 1, \;\; \phi_k(x) = (x - x_0) \cdots (x - x_{k-1}), \; k = 1, \ldots, n.$$

For this basis the polynomial takes the form

$$p(x) = c_0 + c_1(x - x_0) + \ldots + c_n(x - x_0) \cdots (x - x_{n-1}).$$

How do we know it is a basis? First, note that $\phi_k(x)$ is a polynomial of degree k, which implies linear independence. (Any polynomial basis such that $\phi_k(x)$ is a polynomial of degree k is called a 'triangular' basis.) Showing that the Newton basis spans the space of polynomials of degree $\leq n$ is left as a homework exercise.

The linear system for the expansion coefficients of the unique solution to the interpolation problem is triangular when using the Newton basis:

$$\begin{aligned}
p(x_0) &= c_0 &&= y_0 \\
p(x_1) &= c_0 + c_1(x_1 - x_0) &&= y_1 \\
\vdots &= \vdots &&= \;\vdots \\
p(x_n) &= c_0 + c_1(x_n - x_0) + \ldots + c_n(x_n - x_0) \cdots (x_n - x_{n-1}) &&= y_n
\end{aligned}$$

Lower triangular systems are easy to solve via forward substitution (in order n^2 operations). A related benefit is that if you get a new data point $\{x_{n+1}, y_{n+1}\}$ you just need to compute one new coefficient. Could still be ill-conditioned; you can check conditioning by looking at the diagonal elements $1, (x_1 - x_0), (x_2 - x_0)(x_2 - x_1), \ldots$. There are methods for permuting the order of the nodes $\{x_i\}$ to improve the conditioning of the lower-triangular matrix (see D&B).

What about the cost to evaluate the solution once you have it? Notice that the expression for the interpolating polynomial can be nested as follows

$$\begin{aligned}
p(x) &= c_0 + c_1(x - x_0) + c_2(x - x_0)(x - x_1) + \ldots \\
&= c_0 + (x - x_0)\left[c_1 + (x - x_1)[c_2 + \ldots + (x - x_{n-2})[c_{n-1} + (x - x_{n-1})c_n]\ldots]\right]
\end{aligned}$$

To evaluate this expression you evaluate from inner to outer; the cost is $\mathcal{O}(n)$ flops. **Summary:**

- Monomial basis: Solve via Vandermonde $\mathcal{O}(n^3)$, (possibly ill-conditioned), evaluate via nested multiplication $\mathcal{O}(n)$.
- Lagrange basis: Solve cost = 0, direct evaluation $\mathcal{O}(n^2)$.
- Newton basis: Solve cost = $\mathcal{O}(n^2)$ (possibly ill-conditioned, but you can check conditioning easily and also mitigate by re-ordering the x_i), evaluation by nested multiplication $\mathcal{O}(n)$.

Lagrange looks appealing here unless you're evaluating at $\mathcal{O}(n)$ or more locations, in which case Newton is best.

5 There is a cheaper (though still n^2) method for computing the coefficients in the Newton basis. It's faster than forward substitution on the lower-triangular system because it uses the special structure of the matrix (not clear to me that it would generalize to a faster algorithm for general lower triangular matrices). As we derive it we will simultaneously consider the interpolation error (which we've heretofore avoided). Assume that the data comes from some function f such that $y_i = f(x_i)$. Write

$$f(x) = p(x) + A_n(x)(x - x_0)\cdots(x - x_n).$$

We can write the remainder this way because we know that $f(x_i) = p(x_i)$. Expand

$$f(x) = c_0 + c_1(x - x_0) + \ldots + A(x)(x - x_0)\cdots(x - x_n).$$

$$f(x_0) = c_0$$

Subtract

$$f(x) - f(x_0) = c_1(x - x_0) + c_2(x - x_0)(x - x_1) + \ldots + A(x)(x - x_0)\cdots(x - x_n).$$

Divide by $(x - x_0)$

$$\frac{f(x) - f(x_0)}{x - x_0} = c_1 + c_2(x - x_1) + \ldots + A(x)(x - x_1)\cdots(x - x_n).$$

Evaluate at $x = x_1$

$$\frac{f(x_1) - f(x_0)}{x_1 - x_0} = c_1.$$

Notice that we can carry on: subtract out c_i, then divide by $(x - x_i)$, then evaluate at x_{i+1}. Define some notation

$$f[x] = f(x), \quad f[\cdots, x_k, x] = \frac{f[\cdots, x] - f[\cdots, x_k]}{x - x_k}$$

E.g.

$$f[x] = f(x), \quad f[x_0, x] = \frac{f[x] - f[x_0]}{x - x_0}, \quad f[x_0, x_1, x] = \frac{f[x_0, x] - f[x_0, x_1]}{x - x_1}, \text{ etc.}$$

These are called **Newton Divided Differences**. Returning to the above derivation,

$$\frac{f(x_1) - f(x_0)}{x_1 - x_0} = c_1 = f[x_0, x_1].$$

$$f[x_0, x] = f[x_0, x_1] + c_2(x - x_1) + \ldots + A(x)(x - x_1)\cdots(x - x_n)$$

Subtract, divide

$$\frac{f[x_0, x] - f[x_0, x_1]}{x - x_1} = f[x_0, x_1, x] = c_2 + c_3(x - x_2) + \ldots + A(x)(x - x_2)\cdots(x - x_n)$$

Evaluate at $x = x_2$:

$$f[x_0, x_1, x_2] = c_2$$

In general

$$c_k = f[x_0, \ldots, x_k], \quad k = 0, \ldots, n$$

and at the last step we have

$$f[x_0, \ldots, x_n, x] = A(x).$$

We've arrived at expressions giving the (i) expansion coefficients c_i and (ii) interpolation error, both in terms of Newton divided differences.

$$f(x) - p(x) = f[x_0, \ldots, x_n, x](x - x_0)\cdots(x - x_n).$$

Note that the interpolating polynomial is unique (of fixed degree), so this error formula applies regardless of which basis you use, which method you use to compute coefficients, etc.

6 How can we compute the NDD? Use a tabular form

$$\begin{array}{cccc} x_0 & f[x_0] & f[x_0,x_1] & f[x_0,x_1,x_2] & \dots \\ x_1 & f[x_1] & f[x_1,x_2] & f[x_1,x_2,x_3] & \dots \end{array}$$

Show that information propagates left to right and bottom to top. To compute any value on the top row, you only need to compute the lower-left triangle below it. Notice that the above algorithm is based on symmetry of the divided difference formula

$$f[x_0,x_1,x_2] = f[x_1,x_0,x_2] = \frac{f[x_1,x_0]-f[x_1,x_2]}{x_0-x_2} = \frac{f[x_0,x_1]-f[x_1,x_2]}{x_0-x_2}.$$

7 Interpolation error without NDD. Consider

$$E(x) = f(x) - p(x)$$

where p is the unique interpolating polynomial of degree $\leq n$. Define auxiliary functions $\Psi(t)$ and $G(t)$

$$\Psi(t) = (t-x_0)\cdots(t-x_n), \;\; G(t) = E(t) - \frac{\Psi(t)}{\Psi(x)}E(x).$$

The idea here is that if you specify distinct points $x_0,\dots,x_n$ and x, then we have an auxiliary function $G(t)$ that depends on all these points.

Now notice that $G(t=x_i)=0$ for any x_i since both $E(x_i)=0$ and $\Psi(x_i)=0$. **Also** $G(t=x)$ is 0. So G has $n+2$ zeros on the interval that contains $x_0,\dots,x_n$ and x.

Consider a continuously-differentiable function that has 2 zeros. There must be some ξ in the open interval such that the derivative at ξ is 0.

Now consider a C^2 function with 3 zeros. First we infer the existence of two points with zero slope, then we infer a single point with second derivative equal to zero.

Rolle's theorem says that for a function G with $n+2$ distinct zeros and $n+1$ continuous derivatives there must be a point ξ in the interval containing the zeros such that $G^{(n+1)}(\xi)=0$. If we assume that f has $n+1$ continuous derivatives then we know

$$0 = G^{(n+1)}(\xi) = f^{(n+1)}(\xi) - p^{(n+1)}(\xi) - \frac{\Psi^{(n+1)}(\xi)}{\Psi(x)}E(x)$$

Notice that $p^{(n+1)}(x)=0$ since it's a polynomial of degree $\leq n$. Also notice that $\Psi^{(n+1)} = (n+1)!$. Then

$$E(x) = \frac{(x-x_0)\cdots(x-x_n)}{(n+1)!}f^{(n+1)}(\xi)$$

for some ξ in the interval containing $x_0,\dots,x_n$ and x.

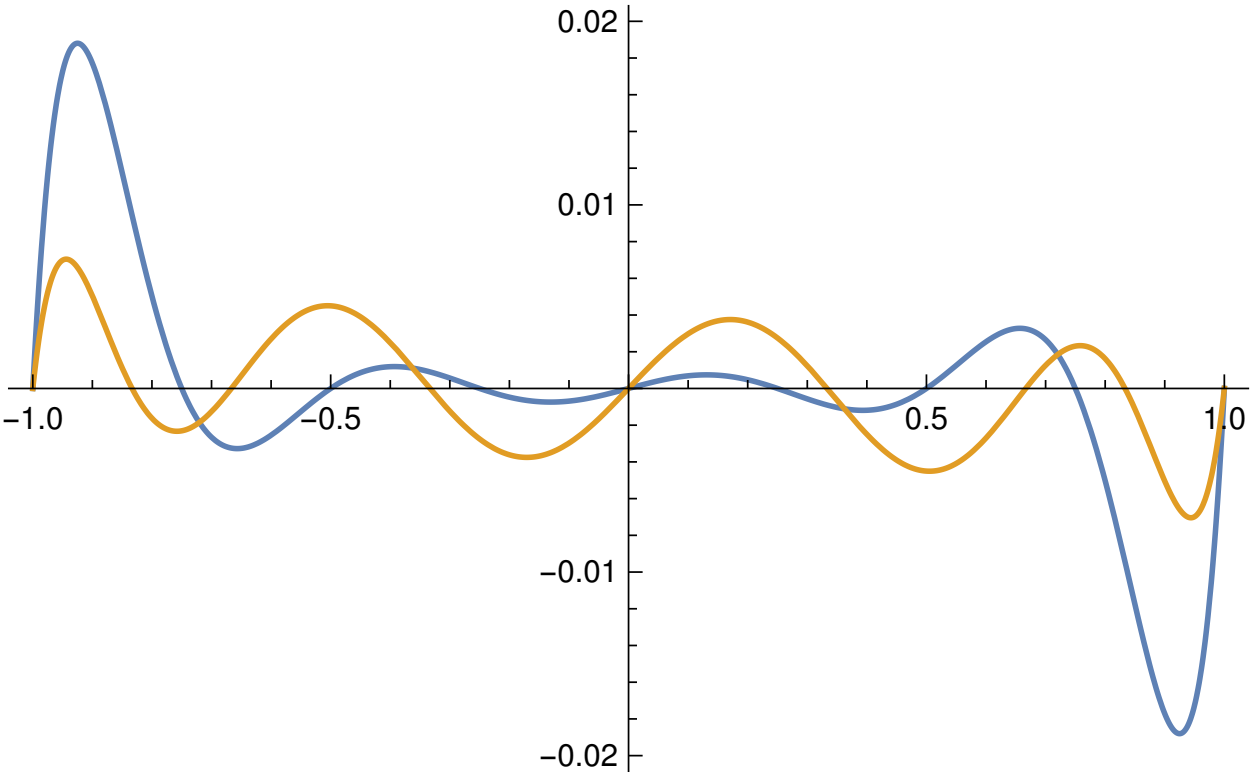
Comparing this to the formula we derived using NDD:

$$f[x_0,\dots,x_n,x] = \frac{f^{(n+1)}(\xi)}{(n+1)!}$$

or

$$f[x_0,\dots,x_m] = \frac{f^{(m)}(\xi)}{m!}$$

where $m=n+1$ and $x=x_{n+1}$ to make the formula cleaner.



8* Runge phenomenon. It is interesting to ask whether the interpolant converges to the true function in the limit of a large number of points, or more generally just how it behaves for large numbers of points.

Note the connection to Taylor's theorem, which also makes polynomial approximations and has a remainder term related to $(x-c)^{n+1}f^{(n+1)}(\xi)/(n+1)!$. Taylor approximation converges exponentially over the open disk whose boundary passes through the nearest singularity of f (in the complex plane). So you can see that the remainder term can grow if there's a singularity.

Unlike Taylor approximation, interpolation is spread over an interval and uses many points. So you have to define a way of specifying the limiting distribution of interpolation points as $n \to \infty$; e.g. equidistant points have a uniform distribution, or you could have a distribution with more points in the center, etc. Interpolation is also different from Taylor in that $f^{(n+1)}(\xi)/(n+1)!$ is multiplied by a different polynomial: $\Psi(x)$ instead of $(x-c)^{n+1}$. Consider the behavior of the maximum of $|\Psi|$ over the interpolation interval as $n \to \infty$. The behavior depends on the location of the nodes.

Carl Runge showed that the equispaced interpolant of $(1+25x^2)^{-1}$ behaves poorly over $[-1,1]$ (try it yourself!), so the phenomenon of non-convergent oscillatory interpolants on equispaced nodes is called the 'Runge phenomenon.' The analysis is complicated (see Dahlquist & Bjork §4.2.6), but it can be shown that if your interpolation points are roots or extrema of the Chebyshev polynomials, then the interpolant will converge to any function that is analytic on the interval.

9 Chebyshev & Interpolation. (Atkinson, 4.12 & related.) We just stated (without proof) that if you interpolate at the roots of the Chebyshev polynomials you will avoid the Runge phenomenon. We will now consider interpolation at the Chebyshev roots from a different perspective. The error in interpolating at $-1 \le x_0 < x_1 < \ldots < x_n \le 1$ is

$$f(x) - p_n(x) = \frac{(x-x_0)\cdots(x-x_n)}{(n+1)!} f^{(n+1)}(\xi)$$

as long as f has the required number of derivatives (otherwise we only have the divided-difference error formula). We can't control the $f^{(n+1)}(\xi)$ part of the error, but let's try to pick the interpolation nodes to

minimize

$$|(x - x_0) \cdots (x - x_n)|.$$

Notice that this is a polynomial of degree $n + 1$ of the form

$$(x - x_0) \cdots (x - x_n) = x^{n+1} + \ldots$$

So let's consider the question in a different perspective:

Find the polynomial $M(x) = x^{n+1} + \ldots$ that minimizes

$$\|M(x)\|_\infty.$$

Before we start, consider that the three-term recursion for Chebyshev polynomials is

$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x), \;\; T_0(x) = 1, \; T_1(x) = x.$$

It's clear that this implies

$$T_{n+1}(x) = 2^n x^{n+1} + \text{polynomial of degree } \leq n.$$

Also, T_{n+1} has exactly $n + 1$ distinct roots in $[-1, 1]$ (we proved this for any OPs). These statements imply that $T_{n+1}(x)/2^n = (x-x_0)(x-x_1)\cdots(x-x_n)$ where x_i are the roots, i.e. it is within the class of polynomials that we're considering. Also recall that $T_{n+1}(x)$, being just a cosine, has $\|T_{n+1}(x)\|_\infty = 1$, so

$$\left\| \frac{T_{n+1}(x)}{2^n} \right\|_\infty = \frac{1}{2^n}.$$

Finally, note that the max is attained at points where

$$T_{n+1}(x) = \cos((n + 1)\cos^{-1}(x)) = \pm 1.$$

Cosine is ± 1 at integer multiples of π, so the points $x \in [-1, 1]$ are those for which $(n + 1)\cos^{-1}(x) = \pm j\pi$ for some integer j, i.e.

$$x_j = \cos\left(\frac{j\pi}{n+1}\right), \;\; j = 0, \ldots, n + 1.$$

Although $T_{n+1}(x)$ has $n + 1$ roots in $[-1, 1]$, it has $n + 2$ extrema.

Now let's return to the question we were considering. We want to construct a polynomial of the form $(x - x_0) \cdots (x - x_n)$ with the smallest possible L^∞ norm (assuming this is possible, i.e. that a minimizer exists). Such a polynomial will either have L^∞ norm less than 2^{-n}, or we might as well just use the Chebyshev polynomial. We will now prove that it's not possible to have a polynomial of the appropriate form with L^∞ norm strictly less than 2^{-n}; the proof will be by contradiction (reductio).

Suppose there is a polynomial $M(x) = x^{n+1} + Q(x)$ (degree of $Q \leq n$) with the property that $\|M(x)\|_\infty < 2^{-n}$, and consider the remainder

$$R(x) = \frac{T_{n+1}(x)}{2^n} - M(x).$$

This is a polynomial of degree at most n. Now consider this polynomial at the extrema x_j of $T_{n+1}(x)$:

$$j = 0 : R(x_0) = \frac{1}{2^n} - M(x_0) > 0$$

It has to be positive since $M(x)$ is, by the assumption we will contradict, everywhere smaller than 2^{-n}.

$$j = 1 : R(x_1) = -\frac{1}{2^n} - M(x_1) < 0$$

$$j = 2 : R(x_2) = \frac{1}{2^n} - M(x_2) > 0$$

etc. There are $n+2$ extrema, so $R(x)$ changes sign $n+1$ times, i.e. has $n+1$ roots in the interval. But $R(x)$ is a polynomial of degree at most n, so this is a contradiction.

The conclusion is that choosing to interpolate at the roots of the Chebyshev polynomials gives us an optimal bound on (part of) the interpolation error

$$|f(x) - p_n(x)| \leq \frac{\max |f^{(n+1)}(x)|}{(n+1)!2^n}.$$

This still doesn't *show* that we avoid the Runge phenomenon when interpolating at the Chebyshev roots, but it does show that interpolating at the Chebyshev roots is in some sense optimal.

10* Gibbs oscillations. We've seen that interpolation can converge for smooth functions if the nodes are carefully chosen. In the other direction you might ask whether some kind of convergence is possible for discontinuous functions. The answer is that yes, sometimes you can get pointwise convergence even for discontinuous functions, but the convergence is not uniform: On either side of a discontinuity there will be oscillations. At any point the oscillations converge to 0 but at any n there is a point near the discontinuity that is affected by the oscillations. For analysis of this phenomenon see Approximation Theory and Approximation Practice, Chapter 9, by Trefethen.

11* Sensitivity (D&B 4.1.3). As usual, we want to know how sensitive the solution to the interpolation problem is to the input data, i.e. the function values $f(x_i) = y_i$. Let $p(x) = \sum_i y_i \ell_i(x)$ be the polynomial interpolating the data $\boldsymbol{y}$. Now perturb the data and define $p_\epsilon(x) = \sum_i (y_i + \epsilon_i)\ell_i(x) = p(x) = \sum_i \epsilon_i \ell_i(x)$. The error between the polynomials is

$$p(x) - p_\epsilon(x) = \sum_i \epsilon_i \ell_i(x).$$

The next question is how to measure the norm of the error. A natural choice is the ∞ norm (e.g. over $x \in [x_0, x_n]$), in which case

$$\|p(x) - p_\epsilon(x)\|_\infty = \max_x |\sum_i \epsilon_i \ell_i(x)| \leq \sum_i |\epsilon_i| \max_x |\ell_i(x)| \leq \left(\max_i |\epsilon_i|\right) \sum_i \max_x |\ell_i(x)| = \Lambda \|\boldsymbol{\epsilon}\|_\infty.$$

The 'Lebesgue constant'

$$\Lambda = \sum_i \max_x |\ell_i(x)|$$

depends only on the location nodes x_i. For equally spaced points the Lebesgue constants grow as

$$\Lambda \sim \frac{2^n}{en\log(n)}.$$

There is a theoretical lower bound (taken over all possible sets of interpolation points) on the Lebesgue constants

$$\Lambda_n \geq \frac{2}{\pi} \ln(n+1) + 0.5215\ldots$$

(See ATAP Chapter 15.) If you use the Chebyshev extrema (defined above) then the Lebesgue constant is bounded above by

$$\lambda_n \leq \frac{2}{\pi} \log(n+1) + 1.$$

(The extrema also avoid Runge phenomenon, just like the Chebyshev roots. Presumably the Lebesgue constants are well-behaved for the Chebyshev roots too, I just don't have a formula.) Of course, you don't always get to choose the interpolation nodes. Summary: Sensitivity of the interpolating polynomial depends on the location of the nodes; equispaced nodes can be really bad; Chebyshev extrema are near-optimal.

12* Sensitivity (D&B §4.1.3 and 4.2.1). Just like with linear systems, the accuracy of our computed solution to the interpolation problem depends both on the conditioning of the exact solution and on the details of the method we use to compute the solution. The analogy here is that a 'good' basis is like a good pivoting strategy in Gaussian elimination. The key part of numerical sensitivity is in computing the expansion coefficients. One method is to use the monomial basis and solve the Vandermonde system. The condition number of the Vandermonde system depends on the location of the nodes. For equidistant points in $[-1, 1]$ and using the ∞ norm,

$$\kappa_\infty(\mathbf{V}) \sim \frac{e^{\pi/4}}{\pi}(3.1)^n \text{ for large } n.$$

This is badly conditioned: for 20 points the condition number is approximately 10^9.

For harmonic points $x_i = 1/(i+1)$ $(i = 1, \ldots, n)$, the condition number is $\kappa_\infty > n^{n+1}$ (faster than exponential). For specially-chosen points you can do better, e.g. for the 'Chebyshev' points in $[-1, 1]$

$$\kappa_\infty(\mathbf{V}) \sim 0.253^{3/4}(1+\sqrt{2})^n.$$

If you use the Newton basis the conditioning depends strongly on the ordering of the interpolation nodes (i.e. you don't have to order them $x_0 < x_1 < \ldots < x_n$). If the standard ordering isn't good enough for you, there are other options to be found in D&B §4.2.1.

Polynomial Hermite Interpolation

1 The Hermite interpolation problem specifies n nodes (note difference compared to $n+1$ nodes in previous section!) $\{x_i\}_1^n$, but now we have multiple data at each node. Simplest case (for exposition) is $\{y_i\}$ and $\{y_i'\}$ at each node, then require

$$p(x_i) = y_i, \qquad p'(x_i) = y_i'.$$

Can be generalized to having different numbers of conditions at each node (see Stoer & Bulirsch). Note connection to Taylor: a single node with lots of derivatives. If you specify something like $p(x_1) = y_1$ and $p''(x_1) = y_1''$, but skip $p'(x_1)$ it's called 'Birkhoff interpolation' or 'lacunary interpolation' and is not always solvable.

There are $2n$ conditions, so we seek a polynomial solution of degree $\leq 2n-1$. The associated linear system is

$$\begin{aligned}
p(x_1) &= a_0 + a_1 x_1 + \ldots + a_{2n-1} x_1^{2n-1} = y_1 \\
&\vdots \\
p(x_n) &= a_0 + a_1 x_n + \ldots + a_{2n-1} x_n^{2n-1} = y_n \\
p'(x_1) &= a_1 + 2a_2 x_1 \ldots + (2n-1) a_{2n-1} x_1^{2n-2} = y_1' \\
&\vdots \\
p'(x_n) &= a_1 + 2a_2 x_n + \ldots + a_{2n-1} x_n^{2n-1} = y_n'
\end{aligned}$$

The matrix is called a 'confluent Vandermonde' matrix.

As previously, we will show that a unique solution exists by construction of a cardinal basis. Define

$$h_i(x) = [1 - 2\ell_i'(x_i)(x - x_i)]\ell_i^2(x)$$

$$\tilde{h}_i(x) = (x - x_i)\ell_i^2(x)$$

Note that the Lagrange polynomials in this case are of degree $n-1$ so both h_i and $\tilde{h}_i$ are of degree $\leq 2n-1$. It is easy to show that $h_i(x_j) = \delta_{ij}$, $h_i'(x_j) = 0$, $\tilde{h}_i(x_j) = 0$, and $\tilde{h}_i'(x_j) = \delta_{ij}$. Using this set of functions, we can write the Hermite interpolating polynomial for any data

$$p(x) = \sum_{i=1}^n y_i h_i(x) + \sum_{i=1}^n y_i' \tilde{h}_i(x)$$

The fact that the confluent Vandermonde system *has* a solution for any data, means that the solution must be unique. If solutions were not unique then the linear system would have a nontrivial cokernel, and it would be possible to construct a RHS for which the system has no solution.

2 NDD for Hermite interpolation. First, note that if we assume f is differentiable and take the limit $f[x,y]$ as $y \to x$

$$\lim_{y\to x} f[x,y] = \lim_{y\to x} \frac{f(y)-f(x)}{y-x} = f'(x)$$

This is in complete agreement with our previous formula

$$f[x,y] = f'(\xi).$$

In general, if f is sufficiently differentiable

$$f[x,\ldots,x] = \frac{f^{(n-1)}(x)}{(n-1)!}$$

where there are n x's in the brackets on the left.

The Newton basis for (this kind of) Hermite interpolation is

$$1,\ (x-x_1),\ (x-x_1)^2,\ (x-x_1)^2(x-x_2),\ \ldots,(x-x_1)^2\cdots(x-x_{n-1})^2(x-x_n)$$

Write

$$f(x) = p(x) + \Psi(x)^2 A(x)$$

Solve for the coefficients in the Newton basis as follows

$$f(x_1) = c_0 = f[x_1]$$

$$f'(x_1) = c_1 = f[x_1,x_1]$$

$$f(x_2) = c_0 + c_1(x_2-x_1) + c_2(x_2-x_1)^2$$

$$c_2 = \frac{\frac{f(x_2)-f(x_1)}{x_2-x_1} - f'(x_1)}{x_2-x_1} = \frac{f[x_1,x_2]-f[x_1,x_1]}{x_2-x_1} = f[x_1,x_1,x_2]$$

etc. The final solution is

$$p(x) = f[x_1]+f[x_1,x_1](x-x_1)+f[x_1,x_1,x_2](x-x_1)^2+\cdots+f[x_1,x_1,\ldots,x_n,x_n](x-x_1)^2\cdots(x-x_{n-1})^2(x-x_n)$$

The error formula is

$$f(x)-p(x) = \Psi(x)^2 f[x_1,x_1,\ldots,x_n,x_n,x].$$

Using the relationship between divided differences and derivatives, and assuming f is sufficiently differentiable, we have

$$f(x)-p(x) = \Psi(x)^2 \frac{f^{(2n)}(\xi)}{(2n)!}$$

for some ξ in the interval containing the interpolation points and x.

Piecewise Polynomial Interpolation

1 Local piecewise polynomial interpolation. Suppose we have $\{x_i\}_0^n$ and $\{y_i\}_0^n$ as usual. Split the data up into sub-groups, where each group has $\{x_i, y_i\}$; groups should overlap only at endpoints, if at all. **Local** piecewise interolation interpolates each sub-group separately. Can be discontinuous or continuous; Hermite can be smooth at the group boundaries, etc.

Consider, for example, piecewise-linear interpolation with an equispaced grid of size $x_{i+1} - x_i = h$. On a single interval

$$p(x) = y_i \frac{(x_{i+1} - x)}{h} + y_{i+1} \frac{(x - x_i)}{h}, \quad x \in [x_i, x_{i+1}].$$

The error on a single interval is

$$E(x) = (x - x_i)(x - x_{i+1}) \frac{f''(\xi)}{2}.$$

The maximum of $|(x - x_i)(x - x_{i+1})|$ occurs halfway through the interval, with value $h^2/4$, so

$$|E(x)| \leq \frac{h^2 |f''(\xi)|}{8}.$$

If $|f''|$ is bounded over the whole interval of interpolation, then by letting $h \to 0$ the approximation will converge. This is a benefit compared to global polynomial interpolation, which can suffer the Runge phenomenon. Making h smaller is called h-refinement, while increasing the degree of the polynomial on each segment is called p-refinement. A benefit of piecewise-linear is that it preserves the range and monotonicity properties of data. The main drawback to local piecewise polynomial interpolation is that the result is not smooth (unless you use Hermite & have the requisite information about f').

2 (This is not on the prelim syllabus, but it is close enough to interpolation that it can show up on the exam.) Finite volume reconstruction. Suppose that instead of point values for data you have nodes $\{x_i\}_{i=0}^n$ and integrals

$$\int_{x_i}^{x_{i+1}} f(x) \mathrm{d}x = \bar{f}_i, \quad i = 0, \ldots, n-1.$$

We now want to find a polynomial $p(x)$ that satisfies conditions

$$\int_{x_i}^{x_{i+1}} p(x) \mathrm{d}x = \bar{f}_i, \quad i = 0, \ldots, n-1.$$

This consistutes a system of n linear constraints on the polynomial, so it is natural to seek a solution in the space of polynomials of degree at most $n-1$. Rather than analyze the problem directly, we will reduce it to a conventional polynomial interpolation problem.

Let

$$F(x) = \int_{x_0}^{x} f(t) \mathrm{d}t$$

and note that we now know conventional pointwise values for $F(x)$:

$$F(x_0) = 0, \quad F(x_k) = \sum_{i=0}^{k-1} \bar{f}_i, \quad k = 1, \ldots, n.$$

Suppose that we just interpolated $F(x)$ by a polynomial $P(x)$ of degree at most n. We know that such a polynomial exists and is unique as long as the nodes don't coincide. Does $P'(x)$ solve the finite volume problem? P solves the interpolation problem, so

$$P(x_k) = \sum_{i=0}^{k-1} \bar{f}_i. \tag{*}$$

The fundamental theorem of calculus tells us

$$P(x_{k+1}) - P(x_k) = \int_{x_k}^{x_{k+1}} P'(t) \mathrm{d}t.$$

Substituting in (*) yields

$$\bar{f}_k = \int_{x_k}^{x_{k+1}} P'(t) \mathrm{d}t$$

so P' solves the finite-volume problem. The fact that the finite volume system is linear and has n constraints means that P' must be the unique polynomial of degree at most $n-1$ that solves the finite volume reconstruction problem.

We now have existence and uniqueness for the finite volume reconstruction problem. What about error? Our interpolation error bound tells us that

$$F(x) - P(x) = \frac{\Psi(x)}{(n+1)!} F^{(n+1)}(\xi(x)) = \frac{\Psi(x)}{(n+1)!} f^{(n)}(\xi(x)).$$

We can convert this to an error bound between $f(x)$ and $P'(x)$ by taking the derivative.

$$f(x) - P'(x) = \frac{\Psi'(x)}{(n+1)!} f^{(n)}(\xi(x)) + \frac{\Psi(x)}{(n+1)!} f^{(n+1)}(\xi(x))\xi'(x).$$

I'm not aware of any theorems saying that ξ is a differentiable function of x, but this derivation is following what I've found in the literature (unsatisfying!).

Now in the homework we saw that if the spacing is uniform $x_{i+1} - x_i = h$ then we can bound the second term by

$$\| \frac{\Psi(x)}{(n+1)!} f^{(n+1)}(\xi(x))\xi'(x) \|_\infty \leq C_0 \frac{h^{n+1}}{n+1}, \;\; C_0 = \| f^{(n+1)}(\xi(x))\xi'(x) \|_\infty$$

(Again, we're just assuming that ξ' is nice!) The other term includes the function

$$\Psi'(x) = \sum_{i=0}^{n} \Pi_{j \neq i} (x - x_j).$$

The same arguments used in the homework to bound $\Psi(x)$ can be applied to each term in this sum, leading to a (pessimistic) bound

$$\| \frac{\Psi'(x)}{(n+1)!} f^{(n)}(\xi(x)) \|_\infty \leq C_1 h^n, \;\; C_1 = \| f^{(n)}(\xi(x)) \|_\infty.$$

Finite volume methods are typically used piecewise, so that the appropriate limit for convergence is $h \to 0$ with n fixed, rather than the 'global' limit of $h \to 0$ and $n \to \infty$. In the piecewise limit the error in the finite-volume reconstruction is dominated by the term $C_1 h^n$, and since n is fixed you usually see that the error decreases as $C_2 h^n$ for some constant C_2 that depends on f and n but not h.

3 Global piecewise polynomial interpolation. The goal here is to find a function that is piecewise-polynomial and whose derivatives up to some order are continuous across segment boundaries. More formally, define a **spline** $s(x)$ of order k associated with the nodes $x_0 < x_1 < \ldots < x_n$

- On each segment $[x_i, x_{i+1}]$ $s(x)$ is a polynomial of degree $< k$
- $s^{(p)}$ is continuous for $p \leq k-2$.

Comments:

- The term 'spline' is related to by-hand drafting where flexible pieces of wood were pressed agains pegs to form a continuous curve.
- Terminology: a 'cubic' spline has polynomials up to cubic, but it's called a fourth-order spline...
- The spline of order 2 is just the piecewise-linear approximation.
- The integral of a spline of order k is a spline of order $k+1$, and the derivative of a spline of order k is a spline of order $k-1$ (barring edge cases).
- The set of splines associated with $\{x_i\}$ is a vector space, and the set of polynomials of degree $\leq k$ is a linear subspace of the set of splines. What is the dimension of the space? We will construct a basis and count the number of basis functions.

Next time do this by starting with a spline of order 1, with a basis of n Heaviside functions. Then integrate this basis to get a spline of order 2, where the basis is integrals of Heaviside functions plus a constant. The integrals of Heaviside functions are the truncated power functions. Show that this is a basis for splines of order 2. Continuing to splines of order k the basis consists of n integrals of Heaviside functions plus $k-1$ monomial terms coming from constants of integration, yielding $k+n-1$ basis functions.

As noted above, the integral of a spline of order k is a spline of order $k+1$, so we'll build a basis for the space of splines of order k by integrating a basis for splines of order $k-1$. Start with splines of order $k=1$, which are piecewise constant. One basis for this space is Heaviside (step) functions

$$\{H(x-x_i)\}_{i=0}^{n-1}.$$

We don't need $H(x-x_n)$ because it's zero on $[x_0, x_n]$. The dimension equals the number of basis functions, which is here n. Now integrate these basis functions and add a constant of integration to get a basis for splines of order $k=2$, i.e. piecewise-linear. The new basis is

$$\{1, (x-x_i)_+\}_{i=0}^{n-1}$$

Define the 'truncated power functions'

$$(x-x_i)_+^j = (\max\{x-x_i, 0\})^j .$$

Notice that they have all derivatives up to $j-1$ continuous and equal to 0 at $x=x_i$. If we keep integrating, the space of splines of order k will have the following basis:

$$\{1, x, \ldots, x^{k-1}, (x-x_1)_+^{k-1}, \ldots, (x-x_{n-1})^{k-1}$$

(cf D&B Theorem 4.4.4). The dimension of the space is therefore k (polynomials) $+n-1$ (truncated power functions) $= k+n-1$. **This is not a good basis for computation! Useful mainly for this proof.**

We next want to find out whether an interpolating spline exists & is unique for any given data. There are $n+1$ interpolation conditions $s(x_i)=y_i$ (this guarantees continuity of the spline). These conditions are linear:

$$s(x_i) = c_0 + c_1 x_i + \ldots + c_{k-1}x_i^{k-1} + c_k(x_i-x_1)_+^{k-1} + \ldots + c_{k+n-1}(x_i - x_{n-1})^{k-1} = y_i$$

(By using this basis we automatically satisfy the continuity conditions on derivatives.) For $k=2$ the number of conditions $n+1$ is the same as the number of unknowns $k+n-1$, but for $k>2$ we need some extra conditions to at least allow a square system with a unique solution. So at this point we've proven neither existence nor uniqueness, but we've at least learned that we're going to need auxiliary conditions for uniqueness.

4 Existence & uniqueness for cubic splines (Atkinson pp 167-ff). For polynomial interpolation & Hermite interpolation we first set up a square linear system for the coefficients, then used a cardinal basis to prove that a solution exists for any data, then referred back to the linear system to get uniqueness. I'm not aware of a simple cardinal basis for splines, and in any case our linear system is not square. So instead we'll specialize to the most common case of cubic splines ($k=4$), and look at auxiliary conditions that guarantee existence & uniqueness.

First note that $s''(x)$ is piecewise-linear for a cubic spline. So, denote $M_i = s''(x_i)$, and we can write

$$s''(x) = \frac{M_i(x_{i+1}-x) + M_{i+1}(x-x_i)}{h_i}, \quad x \in [x_i, x_{i+1}].$$

At this point the M_i are unknown. Now integrate twice on each segment

$$s(x) = \frac{M_i(x_{i+1}-x)^3 + M_{i+1}(x-x_i)^3}{6h_i} + \alpha_i x + \beta_i, \quad x \in [x_i, x_{i+1}].$$

Re-write for convenience as

$$s(x) = \frac{M_i(x_{i+1} - x)^3 + M_{i+1}(x - x_i)^3}{6h_i} + C_i(x_{i+1} - x) + D_i(x - x_i), \quad x \in [x_i, x_{i+1}].$$

Now apply the interpolation conditions $s(x_i) = y_i$. This boils down to

$$C_i = \frac{y_i}{h_i} - \frac{h_i M_i}{6}, \quad D_i = \frac{y_{i+1}}{h_i} - \frac{h_i M_{i+1}}{6}$$

At this point $s(x)$ is a cubic polynomial on each segment, it has a continuous second derivative, and it interpolates the data. The last condition that we apply is continuity of the first derivative. Take the derivative over the segments $[x_{i-1}, x_i]$ and $[x_i, x_{i+1}]$, evaluate the expressions at $x = x_i$ and set them equal to each other. After some simplification

$$\frac{h_{i-1}}{6} M_{i-1} + \frac{h_{i-1} + h_i}{3} M_i + \frac{h_i}{6} M_{i+1} = \frac{y_{i+1} - y_i}{h_i} - \frac{y_i - y_{i-1}}{h_{i-1}}, \quad i = 1, \ldots, n-1.$$

We now have a system of $n-1$ equations in $n+1$ unknowns (the M_i are the unknowns). We consider a few common auxiliary conditions

- If you know the derivatives of the function at the endpoints you can apply that condition $s'(x_0) = y'_0$ & $s'(x_n) = y'_n$. This is called the 'complete cubic spline.' It leads to two extra equations:

  $$\frac{h_0}{3} M_0 + \frac{h_0}{6} M_1 = \frac{y_1 - y_0}{h_0} - y'_0$$

  $$\frac{h_{n-1}}{6} M_{n-1} + \frac{h_{n-1}}{3} M_n = y'_n - \frac{y_n - y_{n-1}}{h_{n-1}}$$

  The whole system is now tridiagonal, symmetric (positive definite), and diagonally dominant, i.e. there is a unique solution. The above analysis with the M_i is essentially just a proof that the complete cubic spline exists and is unique.

- You can make the first and second derivatives periodic at the endpoints. You can again prove that this condition leads to existence and uniqueness by directly analyzing the corresponding $n+1 \times n+1$ linear system.

- Require $s^{(3)}(x)$ to be continuous at x_1 and x_{n-1}. This is the 'not a knot' condition. It leads to a nonsingular system.

- Set $M_0 = M_n = 0$ ($s''(x) = 0$ at the endpoints). This is called the 'natural spline' because it's what a piece of wood (historical spline) would do at the endpoints. In this case the resulting system for $M_1, \ldots, M_{n-1}$ is diagonally-dominant, so it is uniquely invertible. Also, the natural spline has the following nice property (no proof):

  > Among all twice-continuously-differentiable functions g that interpolate the data, the natural spline function minimizes $\int_{x_0}^{x_n} (g''(x))\mathrm{d}x$.

  Atkinson (pp 170) incorrectly lists this as a property of the complete cubic spline.

5* We've now looked at existence & uniqueness for cubic splines with certain auxiliary conditions. Next we look at the error in the approximation. Unfortunately the error analysis of cubic splines is a bit too complex for this course, so I'll simply quote some results

- If f is $C^4([a, b])$ then the 'complete' and 'not-a-knot' cubic splines converge to the true function with $\|f - s\|_\infty \leq ch^4 \max |f^{(4)}(x)|$. In fact the first 3 derivatives of s converge to those of f, though with increasing slowness. Beatson, SJNA 1986.

- If f is $C^4([a,b])$, the natural cubic spline converges with order h^2 on $[x_0, x_n]$, but with order h^4 on any compact subset of (x_0, x_n). The errors are related to the fact that the function f might not satisfy the same 'natural' boundary conditions. Dahlquist & Bjork.

6 (B splines are technically on the prelim syllabus.) Computing with splines. The simplest way to compute with splines is just to treat the spline as a piecewise polynomial. Let

$$s(x) = a_k + b_k x + c_k x^2 + d_k x^3 \text{ for } x \in [x_{k-1}, x_k).$$

Solve for the coefficients $\{a_k, b_k, c_k, d_k\}$ by setting up a linear system enforcing the following constraints: (i) interpolation, (ii) continuity of the first derivative, (iii) continuity of the second derivative, and (iv) whichever auxiliary conditions you choose. Then, once the coefficients are available, evaluate $s(x)$ by figuring out which interval x lies in, then evaluating the appropriate cubic. This is a good way to *work with* splines, but it is not a good way to *analyze* splines. The analysis above with the M_i and the truncated power functions is a good way to *analyze* splines, but not a good way to *work with* splines.

B-splines (Atkinson pp 173–176) are a compactly-supported basis for splines. They are not a cardinal basis. They are positive and supported on a bounded interval. For piecewise-linear we have the hat function (which is a cardinal basis)

$$B_i(x) = \begin{cases} 0 & x \notin [x_{i-1}, x_{i+1}] \\ \frac{x - x_{i-1}}{x_i - x_{i-1}} & x \in [x_{i-1}, x_i] \\ \frac{x_{i+1} - x}{x_{i+1} - x_i} & x \in [x_i, x_{i+1}] \end{cases}$$

To get a basis you formally add one node x_{-1} anywhere to the left of x_0 and one node x_{n+1} anywhere to the right of x_n. For piecewise-quadratic there are three segments where the function is nonzero; on the left it is concave up, then concave down, then concave up again. You formally add two nodes left of x_0 and two to the right of x_n. Etc. We will not analyze or use this basis. One benefit of this basis is that the linear system for the coordinates of the interpolating spline is sparse (banded) because the basis functions are compactly supported. **B splines are not a different kind of spline.** They're just a basis that can be convenient for computations (you can read the reams of interesting analytical properties of B splines in Atkinson and D&B).

Trigonometric Interpolation

1 We now consider the trigonometric interpolation problem. We seek for a function of the following form

$$p(x) = a_0 + \sum_{j=1}^{n} a_j \cos(jx) + b_j \sin(jx).$$

If a_n and b_n are not both zero, then this is called a trigonometric polynomial of degree n.

We've implicitly given a basis for our function space (dimension $2n+1$)

$$\{1, \cos(x), \sin(x), \ldots, \cos(nx), \sin(nx)\}.$$

We should really first consider whether these functions are linearly independent; otherwise the interpolation problem posed here might have multiple solutions (if it has any). I will assume you already know that these functions are a basis for their span.

We will reduce the trig interpolation problem to the polynomial one. Note that

$$e^{\mathrm{i}jx} = \cos(jx) + \mathrm{i}\sin(jx)$$

$$\cos(jx) = \frac{e^{\mathrm{i}jx} + e^{-\mathrm{i}jx}}{2}, \qquad \sin(jx) = \frac{e^{\mathrm{i}jx} - e^{-\mathrm{i}jx}}{2\mathrm{i}}$$

so

$$p(x) = \sum_{j=-n}^{n} \tilde{c}_j e^{\mathrm{i}jx} = \sum_{j=-n}^{n} \tilde{c}_j \left(e^{\mathrm{i}x}\right)^j$$

$$\tilde{c}_0 = a_0, \;\; \tilde{c}_j = \frac{1}{2}(a_j + \mathrm{i}b_j), \;\; \tilde{c}_{-j} = \tilde{c}_j^* \equiv a_j = \tilde{c}_j + \tilde{c}_{-j}, \;\; b_j = \frac{\tilde{c}_j - \tilde{c}_{-j}}{\mathrm{i}}$$

$$\tilde{p}(z) = \sum_{j=-n}^{n} \tilde{c}_j z^j, \;\; p(x) = \tilde{p}(z) \text{ when } z = e^{\mathrm{i}x}.$$

$$P(z) = z^n \tilde{p}(z) = \sum_{j=0}^{2n} c_j z^j, \;\; \tilde{c}_{j-n} = c_j$$

P is a polynomial of degree $\leq 2n$. Because there is a one-to-one relationship between the c_j and the a_j, b_j, the trig interpolation problem is equivalent a (complex) polynomial interpolation problem. The latter has a unique solution whenever the z_j are distinct; what does this mean for the x_j? Since $z = e^{\mathrm{i}x}$, the z_j will be distinct whenever the x_j are distinct modulo 2π. Typically we would first rescale all of our x_j so that they lie in the interval $[0, 2\pi)$. Then the trig interpolation problem will have a solution whenever the (rescaled) x_j are distinct.

2 The above shows that the trig interpolation problem has a unique solution provided that the x_j are distinct modulo 2π; what about computing the solution? For non-equispaced nodes you can just solve the associated polynomial problem using Newton Divided Differences, and then re-map the coefficients c_j to the a_j, b_j. If the nodes are equispaced there are better methods.

WLOG, let (not the same as Atkinson)

$$[0, 2\pi) \ni x_j = j\frac{2\pi}{2n+1}, \;\; z_j = e^{\mathrm{i}x_j}, \;\; j = 0, \ldots, 2n.$$

Consider the Vandermonde matrix of the associated polynomial interpolation problem.

$$P(z_j) = \sum_{k=0}^{2n} c_k z_j^k = c_0 + c_1 z_j + \ldots + c_{2n} z_j^{2n} = z_j^n y_j$$

$$\begin{bmatrix} 1 & z_0 & \cdots & z_0^{2n} \\ 1 & z_1 & \cdots & z_1^{2n} \\ & & \vdots & \\ 1 & z_{2n} & \cdots & z_{2n}^{2n} \end{bmatrix}.$$

We will now show that the columns of this matrix are orthogonal with respect to the standard complex dot product. A single column has the form

$$\mathbf{V}_k = \boldsymbol{v}^{(k)}; \;\; \boldsymbol{v}_j^{(k)} = z_j^k, \;\; j = 0, \ldots, 2n, \;\; k = 0, \ldots, 2n.$$

The complex dot product of two columns is

$$\boldsymbol{v}^{(k)} \cdot \boldsymbol{v}^{(l)} = \sum_{j=0}^{2n} z_j^{l-k}$$

Now use

$$z_j^k = \exp\{jk\frac{2\pi\mathrm{i}}{2n+1}\}$$

$$\boldsymbol{v}^{(k)} \cdot \boldsymbol{v}^{(l)} = \sum_j \exp\{j\frac{2\pi\mathrm{i}}{2n+1}(l-k)\} = \sum_{j=0}^{2n} \omega(l-k)^j, \;\; \omega(l-k) = \exp\{(l-k)\frac{2\pi\mathrm{i}}{2n+1}\}$$

We can use the geometric series formula

$$\sum_{j=0}^{2n} \omega(l-k)^j = \frac{1 - \omega(l-k)^{2n+1}}{1 - \omega(l-k)}.$$

Now consider

$$\omega(l-k)^{2n+1} = \exp\{(l-k)\frac{2\pi(2n+1)\mathrm{i}}{2n+1}\} = 1.$$

If $k-l=0$ we just have

$$\boldsymbol{v}^{(k)} \cdot \boldsymbol{v}^{(k)} = 2n+1.$$

The Vandermonde system can be solved easily

$$\mathbf{V}\boldsymbol{c} = \boldsymbol{y}, \quad \boldsymbol{c} = \frac{\mathbf{V}^*\boldsymbol{y}}{2n+1}.$$

The matrix $\mathbf{V}^*$, up to normalization, is the 'discrete Fourier transform.' There are a fast algorithms to apply it to vectors (i.e. costing less than $\mathcal{O}(n^2)$ flops) called the Fast Fourier Transform.

3 FFT (Bjork, Numerical Methods in Matrix Computations §1.8.5). Suppose we have N points $x_k = 2\pi k/N \in [0, 2\pi)$, $k = 0, \ldots, N-1$, we have $f(x_k)$, and we want to find coefficients c_j such that

$$\sum_{j=0}^{N-1} c_j e^{\mathrm{i}jx_k} = f(x_k), \quad k = 0, \ldots, N-1$$

i.e. we are doing Fourier interpolation. The linear system for c_j has the form

$$\begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & e^{2\pi\mathrm{i}/N} & \cdots & (e^{2\pi\mathrm{i}/N})^{N-1} \\ 1 & e^{4\pi\mathrm{i}/N} & \cdots & (e^{4\pi\mathrm{i}/N})^{N-1} \\ \vdots & & & \vdots \\ 1 & e^{2\pi\mathrm{i}(N-1)/N} & \cdots & (e^{2\pi\mathrm{i}(N-1)/N})^{N-1} \end{bmatrix} \boldsymbol{c} = \boldsymbol{f}.$$

The matrix is unitary (up to a factor of N), so we don't have to do Gaussian Elimination to solve; just multiply by the complex-conjugate transpose:

$$\boldsymbol{c} = \frac{1}{N} \begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & e^{-2\pi\mathrm{i}/N} & \cdots & (e^{-2\pi\mathrm{i}/N})^{N-1} \\ 1 & e^{-4\pi\mathrm{i}/N} & \cdots & (e^{-4\pi\mathrm{i}/N})^{N-1} \\ \vdots & & & \vdots \\ 1 & e^{-2\pi\mathrm{i}(N-1)/N} & \cdots & (e^{-2\pi\mathrm{i}(N-1)/N})^{N-1} \end{bmatrix} \boldsymbol{f}.$$

Define the N^{th} root of unity $\omega_N = e^{-2\pi\mathrm{i}/N}$

$$\boldsymbol{c} = \frac{1}{N} \begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & \omega_N & \cdots & \omega_N^{N-1} \\ 1 & \omega_N^2 & \cdots & \omega_N^{2(N-1)} \\ \vdots & & & \vdots \\ 1 & \omega_N^{N-1} & \cdots & \omega_N^{(N-1)^2} \end{bmatrix} \boldsymbol{f}.$$

Define the matrix

$$\mathbf{F}_N = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & \omega_N & \cdots & \omega_N^{N-1} \\ 1 & \omega_N^2 & \cdots & \omega_N^{2(N-1)} \\ \vdots & & & \vdots \\ 1 & \omega_N^{N-1} & \cdots & \omega_N^{(N-1)^2} \end{bmatrix}, \quad (\mathbf{F}_N)_{jk} = \omega_N^{jk} \text{ note the unusual indexing notation convention.}$$

Suppose that $N = 2m$ and apply a permutation $\mathbf{P}_N$ that groups even indices first followed by odd indices. Then

$$\mathbf{F}_N \mathbf{P}_N^T = \left[\begin{array}{c|c} \mathbf{I} & \mathbf{\Omega}_m \\ \hline \mathbf{I} & -\mathbf{\Omega}_m \end{array}\right] \left[\begin{array}{c|c} \mathbf{F}_m & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{F}_m \end{array}\right] \text{ where } \mathbf{\Omega}_m = \text{diag}(1, \omega_N, \ldots, \omega_N^{m-1}).$$

The proof is based on manipulating the roots of unity and using $\omega_N^2 = \omega_m$. To multiply a vector by $\mathbf{F}_N$ we now have to (i) apply a permutation, (ii) compute two half-sized DFTs, and (iii) perform N multiplications and N additions. The standard cost of matrix/vector multiplication is $\mathcal{O}(N^2)$, and this reduces it to two matrix multiplications at cost $\mathcal{O}((N/2)^2)$ plus some $\mathcal{O}(N)$ operations. The idea can be applied recursively in step (ii), and the overall asymptotic cost reduces to $\mathcal{O}(N \log_2(N))$. For a Fourier transform of size $2^{20} \approx 10^6$, the cost of the FFT is approximately 84,000 times less than the standard method. FFT algorithms exist for matrices whose sizes are not powers of 2 and even for matrices with prime size. The algorithms are fastest when the size of the matrix is in powers of small prime numbers like 2, 3, or 5.

Multivariate Interpolation

1 (Dahlquist & Bjork) The multivariate interpolation problem is an obvious extension. Find $p(x)$ s.t. $p(\boldsymbol{x}_i) = f_i$ for $i = 0, \ldots, n$, where $\boldsymbol{x}_i \in \mathbb{R}^d$. As usual, there are infinitely-many solutions, unless we appropriately restrict the space wherein we seek a solution. Consider a 3-point interpolation problem in 2D $\{(x_i, y_i)\}_1^3$, $\{f_i\}_1^3$. Seek a solution of the form $p(x, y) = c_1 + c_2 x + c_3 y$. The linear system for the coefficients is

$$\begin{bmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \end{bmatrix} \begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ f_3 \end{pmatrix}.$$

This is **not** a Vandermonde matrix. Note that if the three points are distinct, but all lie on a straight line (rather than forming a triangle)

$$\boldsymbol{x}_i = \boldsymbol{r}(t_i) = \boldsymbol{r}_0 + t_i \boldsymbol{v}, \;\; t_i \neq t_j$$

the matrix is not invertible. This demonstrates that in general, having distinct nodes is not enough to guarantee existence for multivariate polynomial interpolation. Of course we could expand our search space to allow the existence of a solution, but then we run the risk of losing uniqueness. In general, polynomial interpolation in many dimensions is tricky.

2 Polynomials still work for a 'tensor product' grid of points. E.g. in 2D consider

$$\{x_i\}_0^m \quad \{y_j\}_0^n, \;\; \{(x_i, y_j)\}_{i,j}, \;\; p(x_i, y_j) = f_{i,j}.$$

(Draw picture.) The points don't have to be equispaced, but they do have to form a rectangular grid. Seek a polynomial solution of the form

$$p(x, y) = \sum_{i,j} c_{i,j} x^i y^j.$$

Notice that at a fixed value of y you just have a 1D polynomial interpolation problem, which has a unique solution under the usual conditions that the x_i are distinct:

$$p(x, y_k) = \sum_i d_{i,k} x^i, \;\; d_{i,k} = \sum_j c_{i,j} y_k^j.$$

Consider solving the 1D interpolation problem at each y_k to find all the $d_{i,k}$. Next, to retrieve the $c_{i,j}$ you have to solve the following systems

$$\sum_j c_{i,j} y_k^j = d_{i,k}.$$

Notice that for each i, this is just a 1D polynomial interpolation problem, which has a solution iff the y_k are distinct. So on a tensor-product grid, 2D polynomial interpolation is just a sequence of 1D interpolation problems $\Rightarrow$ a unique solution exists iff the nodes are distinct.

There's a Lagrange cardinal basis for tensor-product grids: just multiply the Lagrange basis for the x and y grid. You can construct a multivariate basis by just taking products of univariate basis functions (which is actually what we did above: our basis $x^i y^j$ is jus the product of the monomial bases in x and y). You can solve the interpolation problem using any of the methods previously discussed, or use specialized algorithms that explicitly account for the higher-dimensional structure. The error formula generalizes in the obvious way. No further discussion here (see Dahlquist & Bjork).

3* You can still solve the multivariate interpolation problem on general 'scattered' points if you look for a solution in a different function space. For example, consider a 'radial' basis of the form

$$\{F_i(\boldsymbol{x}) = \phi(\|\boldsymbol{x} - \boldsymbol{x}_i\|)\}.$$

We want to find a function of the form

$$s(x) = \sum_i c_i F_i(\boldsymbol{x}) = \sum_i c_i \phi(\|\boldsymbol{x} - \boldsymbol{x}_i\|)$$

that solves the interpolation problem, i.e.

$$s(x_j) = \sum_i c_i \phi(\|\boldsymbol{x}_j - \boldsymbol{x}_i\|) = f_i.$$

The coefficient matrix has the form

$$\begin{bmatrix} \phi(0) & \phi(\|\boldsymbol{x}_1 - \boldsymbol{x}_0\|) & \cdots & \phi(\|\boldsymbol{x}_n - \boldsymbol{x}_0\|) \\ \phi(\|\boldsymbol{x}_0 - \boldsymbol{x}_1\|) & \phi(0) & \cdots & \phi(\|\boldsymbol{x}_n - \boldsymbol{x}_1\|) \\ \vdots & & \ddots & \vdots \\ \phi(\|\boldsymbol{x}_0 - \boldsymbol{x}_n\|) & \cdots & \cdots & \phi(0) \end{bmatrix}$$

It's symmetric because $\|\cdot\|$ is symmetric. We still don't know if it's invertible, and the answer depends on the underlying function ϕ. There's a nice theorem called 'Bochner's theorem' which states that the above coefficient matrix is SPD (equivalently a covariance matrix) iff the function ϕ has a non-negative Fourier transform. This theorem is fundamental to spatial statistics; I think there's a proof in Yaglom's "Correlation Theory of Stationary and Related Random Functions: Basic Results." There are lots of these functions, e.g. $\phi(d) = e^{-d^2/\ell^2}$. This example shows that there's a free parameter ℓ in the RBF, which is typical. It controls the condition number of the matrix, as well as the accuracy of the approximation. See Bengt's course on RBFs for further information.