LECTURE 2: *Superior Bases for Polynomial Interpolants*

1.3 *Polynomial interpolants in a general basis*

THE MONOMIAL BASIS may seem like the most natural way to write down the interpolating polynomial, but it can lead to numerical problems, as seen in the previous lecture. To arrive at more stable expressions for the interpolating polynomial, we will derive several different bases for $\mathcal{P}_n$ that give superior computational properties: the expansion coefficients $\{c_j\}$ will typically be smaller, and it will be simpler to determine those coefficients. This is an instance of a general principle of applied mathematics: to promote stability, express your problem in a well-conditioned basis.

Suppose we have some *basis* $\{b_j\}_{j=0}^n$ for $\mathcal{P}_n$. We seek the polynomial $p \in \mathcal{P}_n$ that interpolates f at $x_0, \ldots, x_n$. Write p in the basis as

$$p(x) = c_0 b_0(x) + c_1 b_1(x) + \cdots + c_n b_n(x).$$

Recall that $\{b_j\}_{j=0}^n$ is a *basis* if the functions *span* $\mathcal{P}_n$ and are *linearly independent*. The first requirement means that for any polynomial $p \in \mathcal{P}_n$ we can find constants $c_0, \ldots, c_n$ such that

$$p = c_0 b_0 + \cdots + c_n b_n,$$

while the second requirement means that if

$$0 = c_0 b_0 + \cdots + c_n b_n$$

then we must have $c_0 = \cdots = c_n = 0$.

We seek the coefficients $c_0, \ldots, c_n$ that express the interpolant p in this basis. The interpolation conditions are

$$\begin{aligned} p(x_0) &= c_0 b_0(x_0) + c_1 b_1(x_0) + \cdots + c_n b_n(x_0) = f(x_0) \\ p(x_1) &= c_0 b_0(x_1) + c_1 b_1(x_1) + \cdots + c_n b_n(x_1) = f(x_1) \\ &\vdots \\ p(x_n) &= c_0 b_0(x_n) + c_1 b_1(x_n) + \cdots + c_n b_n(x_n) = f(x_n). \end{aligned}$$

Again we have $n+1$ equations that are linear in the $n+1$ unknowns $c_0, \ldots, c_n$, hence we can arrange these in the matrix form

$$\begin{bmatrix} b_0(x_0) & b_1(x_0) & \cdots & b_n(x_0) \\ b_0(x_1) & b_1(x_1) & \cdots & b_n(x_1) \\ \vdots & \vdots & \ddots & \vdots \\ b_0(x_n) & b_1(x_n) & \cdots & b_n(x_n) \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} f(x_0) \\ f(x_1) \\ \vdots \\ f(x_n) \end{bmatrix}, \tag{1.3}$$

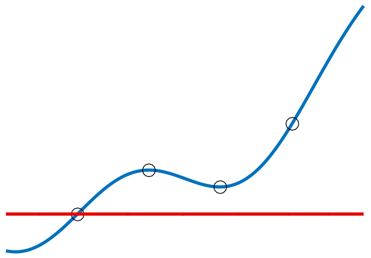
which can be solved via Gaussian elimination for $c_0, \ldots, c_n$.

Notice that the linear system for the monomial basis in (1.1) is a special case of the system in (1.3), with the choice $b_j(x) = x^j$. Next we will look at two superior bases that give more stable expressions for the interpolant. We emphasize that when the basis changes, so to do the values of $c_0, \ldots, c_n$, *but the interpolating polynomial p remains the same, regardless of the basis we use to express it.*

1.4 *Constructing interpolants in the Newton basis*

To derive our first new basis for $\mathcal{P}_n$, we describe an alternative method for constructing the polynomial $p_n \in \mathcal{P}_n$ that interpolates $f \in C[a,b]$ at the distinct points $\{x_0,\ldots,x_n\} \subset [a,b]$. This approach, called the *Newton form* of the interpolant, builds p_n up from lower degree polynomials that interpolate f at only some of the data points.

Begin by constructing the polynomial $p_0 \in \mathcal{P}_0$ that interpolates f at x_0: $p_0(x_0) = f(x_0)$. Since p_0 is a zero-degree polynomial (i.e., a constant), it has the simple form



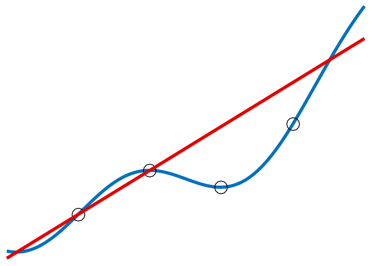
$$p_0(x) = c_0.$$

To satisfy the interpolation condition at x_0, set $c_0 = f(x_0)$. (We emphasize again: this c_0, and the c_j below, will be different from the c_j's obtained in Section 1.2 for the monomial basis.)

Next, use p_0 to build the polynomial $p_1 \in \mathcal{P}_1$ that interpolates f at both x_0 and x_1. In particular, we will require p_1 to have the form

$$p_1(x) = p_0(x) + c_1 q_1(x)$$

for some constant c_1 and some $q_1 \in \mathcal{P}_1$. Note that



$$\begin{aligned} p_1(x_0) &= p_0(x_0) + c_1 q_1(x_0) \\ &= f(x_0) + c_1 q_1(x_0). \end{aligned}$$

Since we require that $p_1(x_0) = f(x_0)$, the above equation implies that $c_1 q_1(x_0) = 0$. Either $c_1 = 0$ (which can only happen in the special case $f(x_0) = f(x_1)$, and we seek a basis that works for *any* f) or $q_1(x_0) = 0$, i.e., $q_1(x_0)$ has a root at x_0. Thus, we deduce that $q_1(x) = x - x_0$. It follows that

$$p_1(x) = c_0 + c_1(x - x_0),$$

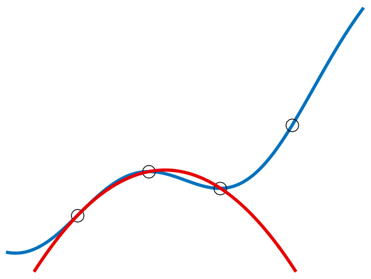
where c_1 is still undetermined. To find c_1, use the interpolation condition at x_1:

$$f(x_1) = p_1(x_1) = c_0 + c_1(x_1 - x_0).$$

Solving for c_1,

$$c_1 = \frac{f(x_1) - c_0}{x_1 - x_0}.$$

Next, find the $p_2 \in \mathcal{P}_2$ that interpolates f at x_0, x_1, and x_2, where p_2 has the form



$$p_2(x) = p_1(x) + c_2 q_2(x).$$

Similar to before, the first term, now $p_1(x)$, 'does the right thing' at the first two interpolation points, $p_1(x_0) = f(x_0)$ and $p_1(x_1) = f(x_1)$.

We require that q_2 not interfere with p_1 at x_0 and x_1, i.e., $q_2(x_0) = q_2(x_1) = 0$. Thus, we take q_2 to have the form

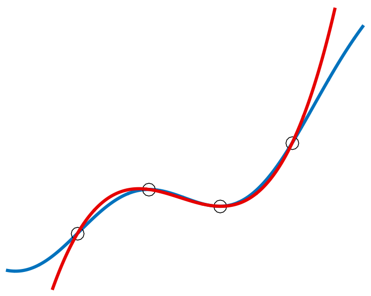
$$q_2(x) = (x - x_0)(x - x_1).$$

The interpolation condition at x_2 gives an equation where c_2 is the only unknown,

$$f(x_2) = p_2(x_2) = p_1(x_2) + c_2 q_2(x_2),$$

which we can solve for

$$c_2 = \frac{f(x_2) - p_1(x_2)}{q_2(x_2)} = \frac{f(x_2) - c_0 - c_1(x_2 - x_0)}{(x_2 - x_0)(x_2 - x_1)}.$$



Follow the same pattern to bootstrap up to p_n, which takes the form

$$p_n(x) = p_{n-1}(x) + c_n q_n(x),$$

where

$$q_n(x) = \prod_{j=0}^{n-1}(x - x_j),$$

and, setting $q_0(x) = 1$, we have

$$c_n = \frac{f(x_n) - \sum_{j=0}^{n-1} c_j q_j(x_n)}{q_n(x_n)}.$$

Finally, the desired polynomial takes the form

$$p_n(x) = \sum_{j=0}^{n} c_j q_j(x).$$

The polynomials q_j for $j = 0, \ldots, n$ form a basis for $\mathcal{P}_n$, called the *Newton basis*. The c_j we have just determined are the expansion coefficients for this interpolant in the Newton basis. Figure 1.5 shows the Newton basis functions q_j for $[a, b] = [0, 1]$ with $n = 5$ and $x_j = j/5$, which look considerably more distinct than the monomial basis polynomials illustrated in Figure 1.1.

This entire procedure for constructing p_n can be condensed into a system of linear equations with the coefficients $\{c_j\}_{j=0}^{n}$ unknown:

$$(1.4) \quad \begin{bmatrix} 1 & & & & \\ 1 & (x_1 - x_0) & & & \\ 1 & (x_2 - x_0) & (x_2 - x_0)(x_2 - x_1) & & \\ \vdots & \vdots & \vdots & \ddots & \\ 1 & (x_n - x_0) & (x_n - x_0)(x_n - x_1) & \cdots & \prod_{j=0}^{n-1}(x_n - x_j) \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} f(x_0) \\ f(x_1) \\ f(x_2) \\ \vdots \\ f(x_n) \end{bmatrix},$$

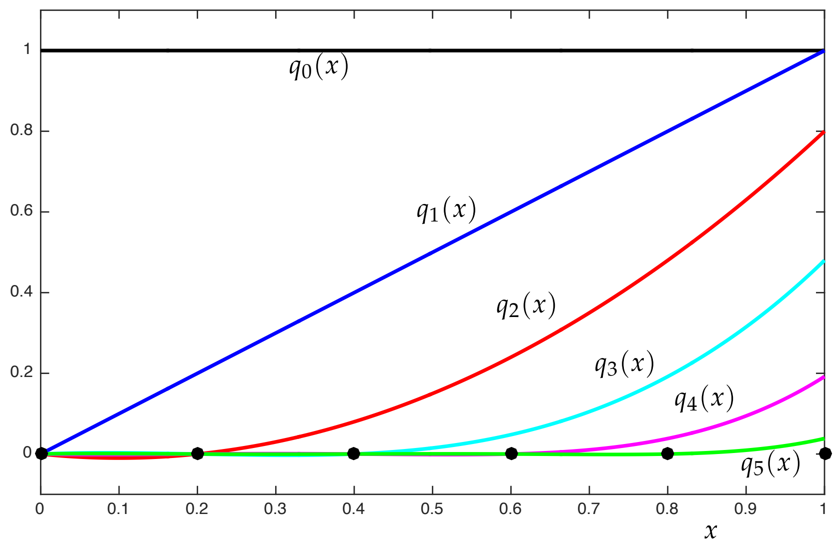


Figure 1.5: The six Newton basis polynomials $q_0, \dots, q_5$ for $\mathcal{P}_5$, based on the interval $[a, b] = [0, 1]$ with $x_j = j/5$ (red circles). Compare these to the monomial basis polynomials in Figure 1.1: these vectors look far more distinct from one another than the monomials.

again a special case of (1.3) but with $b_j(x) = q_j(x)$. (The unspecified entries above the diagonal are zero, since $q_j(x_k) = 0$ when $k < j$.) The system (1.4) involves a triangular matrix, which is simple to solve. Clearly $c_0 = f(x_0)$, and once we know c_0, we can solve for

$$c_1 = \frac{f(x_1) - c_0}{x_1 - x_0}.$$

With c_0 and c_1, we can solve for c_2, and so on. This procedure, *forward substitution*, requires roughly n^2 floating point operations once the entries are formed.

With this Newton form of the interpolant, one can easily update p_n to p_{n+1} in order to incorporate a new data point $(x_{n+1}, f(x_{n+1}))$, as such a change affects neither the previous values of c_j nor q_j. The new data $(x_{n+1}, f(x_{n+1}))$ simply adds a new row to the bottom of the matrix in (1.4), which preserves the triangular structure of the matrix and the values of $\{c_0, \dots, c_n\}$. If we have already found these coefficients, we easily obtain c_{n+1} through one more step of forward substitution.

1.5 *Constructing interpolants in the Lagrange basis*

The monomial basis gave us a linear system (1.1) of the form $\mathbf{Ac} = \mathbf{f}$ in which $\mathbf{A}$ was a *dense* matrix: all of its entries are nonzero. The Newton basis gave a simpler system (1.4) in which $\mathbf{A}$ was a *lower triangular* matrix. Can we go one step further, and find a set of basis functions for which the matrix in (1.3) is *diagonal*?

For the matrix to be diagonal, the jth basis function would need to have roots at all the other interpolation points x_k for $k \neq j$. Such functions, denoted ℓ_j for $j = 0, \ldots, n$, are called *Lagrange basis polynomials*, and they result in the *Lagrange form* of the interpolating polynomial.

We seek to construct $\ell_j \in \mathcal{P}_n$ with $\ell_j(x_k) = 0$ if $j \neq k$, but $\ell_j(x_k) = 1$ if $j = k$. That is, ℓ_j takes the value one at x_j and has roots at all the other n interpolation points.

What form do these basis functions $\ell_j \in \mathcal{P}_n$ take? Since ℓ_j is a degree-n polynomial with the n roots $\{x_k\}_{k=0,k\neq j}^{n}$, it can be written in the form

$$\ell_j(x) = \prod_{k=0,k\neq j}^{n} \gamma_k (x - x_k)$$

for appropriate constants γ_k. We can force $\ell_j(x_j) = 1$ if all the terms in the above product are one when $x = x_j$, i.e., when $\gamma_k = 1/(x_j - x_k)$, so that

$$\ell_j(x) = \prod_{k=0,k\neq j}^{n} \frac{x - x_k}{x_j - x_k}.$$

This form makes it clear that $\ell_j(x_j) = 1$. With these new basis functions, the constants $\{c_j\}$ can be written down immediately. The interpolating polynomial has the form

$$p_n(x) = \sum_{k=0}^{n} c_k \ell_k(x).$$

When $x = x_j$, all terms in this sum will be zero except for one, the $k = j$ term (since $\ell_k(x_j) = 0$ except when $j = k$). Thus,

$$p_n(x_j) = c_j \ell_j(x_j) = c_j,$$

so we can directly write down the coefficients, $c_j = f(x_j)$.

As desired, this approach to constructing basis polynomials leads to a diagonal matrix $\mathbf{A}$ in the equation $\mathbf{Ac} = \mathbf{f}$ for the coefficients. Since we also insisted that $\ell_j(x_j) = 1$, the matrix $\mathbf{A}$ is actually just the *identity* matrix:

$$\begin{bmatrix} 1 & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} f(x_0) \\ f(x_1) \\ \vdots \\ f(x_n) \end{bmatrix}.$$

Now the coefficient matrix is simply the identity.

A forthcoming exercise will investigate an important flexible and numerically stable method for constructing and evaluating Lagrange interpolants known as *barycentric interpolation*.

Figure 1.6 shows the Lagrange basis functions for $n = 5$ with $[a, b] = [0, 1]$ and $x_j = j/5$, the same parameters used in the plots of the monomial and Newton bases earlier.

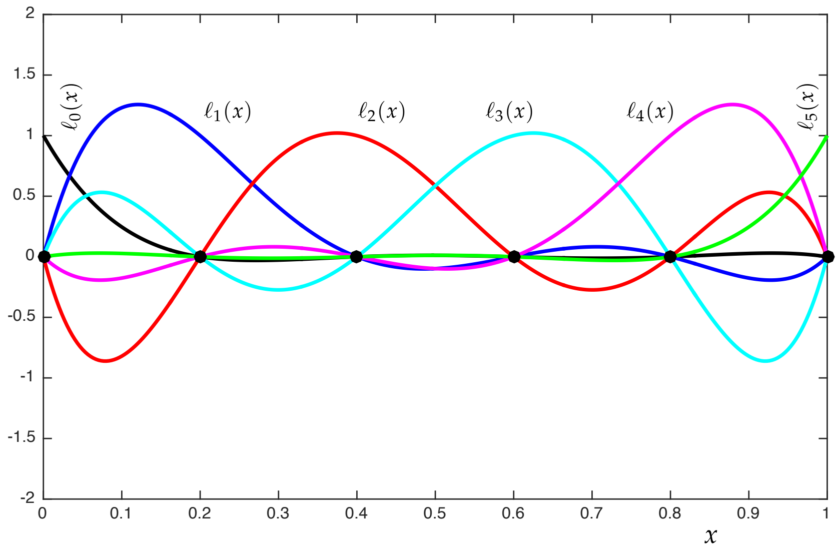


Figure 1.6: The six Lagrange basis polynomials $\ell_0, \ldots, \ell_5$ for $\mathcal{P}_5$, based on the interval $[a, b] = [0, 1]$ with $x_j = j/5$ (red circles). Note that each Lagrange polynomial has roots at n of the interpolation points. Compare these polynomials to the monomial and Newton basis polynomials in Figures 1.1 and 1.5 (but note the different vertical scale): these basis vectors look most independent of all.

The fact that these basis functions are not as closely aligned as the previous ones has interesting consequences on the size of the coefficients $\{c_j\}$. For example, if we have $n + 1 = 6$ interpolation points for $f(x) = \sin(10x) + \cos(10x)$ on $[0, 1]$, we obtain the following coefficients:

	monomial	Newton	Lagrange
c_0	1.0000000e+00	1.0000000e+00	1.0000000e+00
c_1	4.0861958e+01	-2.5342470e+00	4.9315059e-01
c_2	-3.8924180e+02	-1.7459341e+01	-1.4104461e+00
c_3	1.0775024e+03	1.1232385e+02	6.8075479e-01
c_4	-1.1683645e+03	-2.9464687e+02	8.4385821e-01
c_5	4.3685881e+02	4.3685881e+02	-1.3830926e+00

We emphasize that all three approaches (in exact arithmetic) must yield the same unique polynomial, but they are expressed in different bases. The behavior in floating point arithmetic varies significantly with the choice of basis; the monomial basis is the clear loser.