Math 6620: Analysis of Numerical Methods, II
A primer on polynomial interpolation and quadrature

Akil Narayan[1]

[1] Department of Mathematics, and Scientific Computing and Imaging (SCI) Institute
University of Utah

Polynomial interpolation

Let's recall some basics about polynomial interpolation on $\mathbb{R}$: Let $x_1, \ldots x_n$ be any distinct nodes in $\mathbb{R}$, and let $f : C(\mathbb{R}; \mathbb{R})$ be given. Our goal is to construct a polynomial $p \in P_{n-1} := \mathrm{span}\{1, x, \ldots, x^{n-1}\}$ such that,

$$p(x_j) = f(x_j), \qquad j \in [n]$$

There are several ways to tackle this problem; one major outcome is that this problem is *unisolvent*: there is a bijection between $(f_j)_{j \in [n]}$ and P_{n-1}.
(In fact this bijection is a linear map.)

Let's recall some basics about polynomial interpolation on $\mathbb{R}$: Let $x_1, \ldots x_n$ be any distinct nodes in $\mathbb{R}$, and let $f : C(\mathbb{R}; \mathbb{R})$ be given. Our goal is to construct a polynomial $p \in P_{n-1} := \mathrm{span}\{1, x, \ldots, x^{n-1}\}$ such that,

$$p(x_j) = f(x_j), \qquad j \in [n]$$

Direct linear algebra:

$$p(x) = \sum_{j=1}^{n} c_j x^{j-1}.$$

Asserting the interpolation conditions yields the linear system:

$$\boldsymbol{V}\boldsymbol{c} = \boldsymbol{f}, \qquad \boldsymbol{c} = (c_1, \ldots, c_n)^T, \qquad \boldsymbol{f} = (f(x_1), \ldots, f(x_n))^T,$$

where $\boldsymbol{V}$ is a *Vandermonde* matrix:

$$(\boldsymbol{V})_{j,k} = x_j^{k-1}, \qquad j, k \in [n].$$

One can show that $\det \boldsymbol{V} \neq 0$:

$$\det \boldsymbol{V} = \prod_{1 \leqslant j < k \leqslant n} (x_k - x_j),$$

establishing unisolvence.
This procedure requires $\mathcal{O}(n^3)$ effort.

Let's recall some basics about polynomial interpolation on $\mathbb{R}$: Let $x_1, \ldots x_n$ be any distinct nodes in $\mathbb{R}$, and let $f : C(\mathbb{R}; \mathbb{R})$ be given. Our goal is to construct a polynomial $p \in P_{n-1} := \mathrm{span}\{1, x, \ldots, x^{n-1}\}$ such that,

$$p(x_j) = f(x_j), \qquad j \in [n]$$

Newton form:

$$p(x) = \sum_{j=1}^{n} c_j \phi_j(x), \qquad \phi_j(x) := \prod_{k=1}^{j-1} (x - x_k)$$

Note that $\phi_j(x_k) = 0$ for $k < j$, so that a direct linear algebraic system is upper-triangular:

- The diagonal of the corresponding linear system matrix contains no zeros: unisolvence
- The c_j are computable in $\mathcal{O}(n^2)$ effort.

Moreover, back-substitution yields an extremely useful fact: the c_j are simply (Newton) *divided differences*:

$$c_j = f[x_1, \ldots, x_j],$$

where $f[\cdot]$ are defined recursively:

$$f[x_j] = f(x_j), \qquad j \in [n]$$

$$f[x_j, x_{j+1}, \ldots, x_{k-1}, x_k] = \frac{f[x_{j+1}, \ldots, x_k] - f[x_j, \ldots, x_{k-1}]}{x_k - x_j}, \qquad 1 \leqslant j < k \leqslant n.$$

These divided differences are typically computed in a triangular array/tableau (which is $\mathcal{O}(n^2)$ effort to construct).

Let's recall some basics about polynomial interpolation on $\mathbb{R}$: Let $x_1, \ldots x_n$ be any distinct nodes in $\mathbb{R}$, and let $f : C(\mathbb{R}; \mathbb{R})$ be given. Our goal is to construct a polynomial $p \in P_{n-1} := \mathrm{span}\{1, x, \ldots, x^{n-1}\}$ such that,

$$p(x_j) = f(x_j), \qquad\qquad j \in [n]$$

Lagrange form:

$$p(x) = \sum_{j=1}^{n} c_j \ell_j(x), \qquad\qquad \ell_j(x) := \frac{\prod_{k \in [n] \setminus \{k\}} (x - x_k)}{\prod_{k \in [n] \setminus \{k\}} (x_j - x_k)}.$$

The functions ℓ_j are called *Lagrange interpolating polynomials*. Of particular importance is that, by construction, they satisfy,

$$\ell_j(x_k) = \delta_{j,k}, \qquad\qquad j, k \in [n].$$

I.e., the corresponding interpolatory linear algebraic system matrix is $\boldsymbol{I}$. Thus, the coefficients are:

$$c_j = f(x_j).$$

I.e., no linear algebra is required at all: computation of the coefficients is $\mathcal{O}(n)$. But *evaluating* the ℓ_j functions requires $\mathcal{O}(n^2)$ effort.

Again, this procedure confirms univsolvence of the interpolation procedure.

Let's recall some basics about polynomial interpolation on $\mathbb{R}$: Let $x_1, \ldots x_n$ be any distinct nodes in $\mathbb{R}$, and let $f : C(\mathbb{R};\mathbb{R})$ be given. Our goal is to construct a polynomial $p \in P_{n-1} := \mathrm{span}\{1, x, \ldots, x^{n-1}\}$ such that,

$$p(x_j) = f(x_j), \qquad\qquad j \in [n]$$

There are some other approaches to constructing interpolants as well. For example:

$$p(x) = \frac{\sum_{j\in[n]} \frac{w_j f_j}{x - x_j}}{\sum_{j\in[n]} \frac{w_j}{x - x_j}}, \qquad f_j = f(x_j) \qquad\qquad w_j = \frac{1}{\prod_{k\in[n]\setminus\{j\}}(x_k - x_j)}.$$

It is indeed true that $p \in P_{n-1}$ with $p(x_j) = f(x_j)$.

This explicitly provides an $\mathcal{O}(n)$ formula for the interpolant, if $\mathcal{O}(n^2)$ effort is expended to compute the weights w_j.

This is called the barycentric (Lagrange) form of the interoplant.

The error in polynomial interpolation

The error in polynomial approximation is well-studied. Here is one *Lagrange* remainder form for the error:

Theorem

Assume $f \in C^n([a,b])$ *where* $\{x_1, \ldots, x_n\} \subset [a,b]$. *With* p *the degree-*$(n-1)$ *polynomial interpolant of* f, *then for any* $x \in [a,b]$, *there exists a* $y(x) \in [a,b]$ *such that:*

$$f(x) - p(x) = \frac{f^{(n)}(y)}{n!} \prod_{j \in [n]} (x - x_j)$$

The error in polynomial interpolation

The error in polynomial approximation is well-studied. Here is one *Lagrange* remainder form for the error:

Theorem

Assume $f \in C^n([a, b])$ where $\{x_1, \ldots, x_n\} \subset [a, b]$. With p the degree-$(n-1)$ polynomial interpolant of f, then for any $x \in [a, b]$, there exists a $y(x) \in [a, b]$ such that:

$$f(x) - p(x) = \frac{f^{(n)}(y)}{n!} \prod_{j \in [n]} (x - x_j)$$

If the nodes are equispaced:

$$x_j = a + (j-1)h, \qquad h = \frac{b-a}{n-1},$$

then

$$\left| \prod_{j \in [n]} (x - x_j) \right| \leqslant \frac{(n-1)!}{4} h^n$$

i.e., this error is $\mathcal{O}(h^n)$.

The error bound on equispaced nodes decays to zero as $h \downarrow 0$. One can prove the following related result:

Theorem

Let $f \in C([a, b])$. Then there exists some triangular array of nodes, $\{x_{j,n}\}_{j \in [n], n \in \mathbb{N}} \subset [a, b]$, with $\{x_{j,n}\}_{j \in [n]}$ distinct, such that the sequence of polynomials $p_n \in P_{n-1}$ that interpolate f on $\{x_j\}_{j \in [n]}$ converge uniformly to f on $[a, b]$.

The error bound on equispaced nodes decays to zero as $h \downarrow 0$. One can prove the following related result:

Theorem

Let $f \in C([a,b])$. Then there exists some triangular array of nodes, $\{x_{j,n}\}_{j\in[n],n\in\mathbb{N}} \subset [a,b]$, with $\{x_{j,n}\}_{j\in[n]}$ distinct, such that the sequence of polynomials $p_n \in P_{n-1}$ that interpolate f on $\{x_j\}_{j\in[n]}$ converge uniformly to f on $[a,b]$.

Unfortunately, since $f^{(n)}$ can be badly behaved, one can always construct adversarial examples.

Theorem

Let $\{x_{j,n}\}_{j\in[n],n\in\mathbb{N}} \subset [a,b]$ be any triangular array of nodes. Then there exists a function $f \in C([a,b])$ such that the sequence of interpolating polynomials p_n diverges from f in the uniform norm on $[a,b]$.

Nevertheless, the form of the Lagrange remainder error suggests that solving the minimization problem,

$$\min_{\{x_j\}_j \subset [a,b]} \prod_{j \in [n]} (x - x_j),$$

should produce a "good" sequence of nodes.

Nevertheless, the form of the Lagrange remainder error suggests that solving the minimization problem,

$$\min_{\{x_j\}_j \subset [a,b]} \prod_{j \in [n]} (x - x_j),$$

should produce a "good" sequence of nodes.

The solution to this minimization problem is the set of *Chebyshev nodes*, and they provide a reasonable set of interpolation points.

Theorem

If f is absolutely continuous on $[a, b]$, then the sequence of interpolating polynomials on the Chebyshev nodes converges to f in the uniform norm.

Finite differences

Interpolation, and in particular the existence of a reasonable error estimate, suggests a new strategy for constructing finite difference formulas that evaluate derivative of $u^{(p)}(x)$ at some point x_j:

1. Decide on a stencil $x_{j-r}, \ldots, x_j, \ldots x_{j+s}$
2. Construct a degree-$(s+r)$ polynomial that interpolates u at the stencil points
3. Take the pth derivative of u
4. Evaluate the pth derivative of the interpolant at x_j.

(You can convince yourself that indeed this will yield an approximation that is a linear combination of u at the stencil points.)

Finite differences

Interpolation, and in particular the existence of a reasonable error estimate, suggests a new strategy for constructing finite difference formulas that evaluate derivative of $u^{(p)}(x)$ at some point x_j:

1. Decide on a stencil $x_{j-r}, \ldots, x_j, \ldots x_{j+s}$
2. Construct a degree-$(s+r)$ polynomial that interpolates u at the stencil points
3. Take the pth derivative of u
4. Evaluate the pth derivative of the interpolant at x_j.

(You can convince yourself that indeed this will yield an approximation that is a linear combination of u at the stencil points.)

Example

Use polynomial interpolation to construct an approximation to the second derivative at x_j using the stencil x_{j-1}, x_j, x_{j+1}, where $x_{j\pm1} = x_j \pm h$ with $h > 0$.

$$p(x) = u_{j-1}\, \ell_{-1}(x) + u_j\, \ell_0(x) + u_{j+1}\, \ell_1(x)$$

$$\ell_{-1}(x) = \frac{(x-x_j)(x-x_j-h)}{(-h)(-2h)} \qquad \ell_1(x) = \frac{(x-x_j+h)(x-x_j)}{(2h)(h)}$$

$$\ell_0(x) = \frac{(x-x_j+h)(x-x_j-h)}{(h)(-h)}$$

$$p''(x_j) = ?$$

$$= u_{j-1}\, l_{-1}''(x_j) + u_j\, l_0''(x_j) + u_{j+1}\, l_1''(x_j)$$

$$= u_{j-1} \cdot \frac{2}{2h^2} + u_j \frac{2}{-h^2} + u_{j+1} \frac{2}{2h^2}$$

$$= \frac{1}{h^2}\left[u_{j-1} - 2u_j + u_{j+1} \right]$$

The previous exercise suggests that polynomial interpolation is doing something similar to eliminating entries in a Taylor series expansion.

The previous exercise suggests that polynomial interpolation is doing something similar to eliminating entries in a Taylor series expansion.

Theorem

On any stencil of points (equidistant or not), the finite difference formula constructed by eliminating Taylor series terms to as high an order as possible is equivalent to the finite difference formula obtained through polynomial interpolation on the stencil.

In practice, it's frequently easier to use Taylor series, but interpolation has its uses. For example, it's conceptually easier to numerically implement.

A second useful task is approximating integrals via sums of point values, i.e., quadrature:

$$\int_a^b f(x)\mathrm{d}x \approx \sum_{j\in[n]} f(x_j)w_j.$$

Here the "unknowns" are the nodes x_j and the quadrature weights w_j.

Like interpolation, we'll mostly consider the nodes as fixed, and our task is to determine "good" weights.

A second useful task is approximating integrals via sums of point values, i.e., quadrature:

$$\int_a^b f(x)\mathrm{d}x \approx \sum_{j\in[n]} f(x_j)w_j.$$

Here the "unknowns" are the nodes x_j and the quadrature weights w_j.

Like interpolation, we'll mostly consider the nodes as fixed, and our task is to determine "good" weights.

Taking inspiration from finite differences: once the nodes are declared, it's probably a reasonble idea to (i) approximate an integrand by forming a polynomial interpolant and (ii) subsequently integrating the interpolant.

The weights resulting from this procedure yield an *interpolatory quadrature rule*.

A second useful task is approximating integrals via sums of point values, i.e., quadrature:

$$\int_a^b f(x)\mathrm{d}x \approx \sum_{j\in[n]} f(x_j)w_j.$$

Here the "unknowns" are the nodes x_j and the quadrature weights w_j.

Like interpolation, we'll mostly consider the nodes as fixed, and our task is to determine "good" weights.

Taking inspiration from finite differences: once the nodes are declared, it's probably a reasonble idea to (i) approximate an integrand by forming a polynomial interpolant and (ii) subsequently integrating the interpolant.

The weights resulting from this procedure yield an *interpolatory quadrature rule*.

An exercise reveals that interpolatory quadrature rules have the following formula for the weights:

$$w_j = \int_a^b \ell_j(x)\mathrm{d}x,$$

where ℓ_j is the Lagrange polynomial centered at x_j on the nodes $\{x_j\}_{j\in[n]}$.

While integrating Lagrange polynomials is an explicit way to compute interpolatory quadrature weights, like forming finite differences, there is an approach that is typically easier by hand:

Suppose we assert that the weights w_j are formed in order to ensure an exact integral on P_{n-1}:

$$\int_a^b x^k \mathrm{d}x = \sum_{j \in [n]} w_j x_j^k, \qquad k = 0, \ldots, n-1.$$

This is a well-posed strategy (in principle) since it forms a size-n linear system for the n unknown weights.

This approach is called *moment matching*. The principle is to form a quadrature rule that exactly integrates polynomials to as high a degree as possible.

Moment matching

While integrating Lagrange polynomials is an explicit way to compute interpolatory quadrature weights, like forming finite differences, there is an approach that is typically easier by hand:

Suppose we assert that the weights w_j are formed in order to ensure an exact integral on P_{n-1}:

$$\int_a^b x^k \mathrm{d}x = \sum_{j\in[n]} w_j x_j^k, \qquad\qquad k = 0, \ldots, n-1.$$

This is a well-posed strategy (in principle) since it forms a size-n linear system for the n unknown weights.

This approach is called *moment matching*. The principle is to form a quadrature rule that exactly integrates polynomials to as high a degree as possible.

Like the finite difference case, there is an equivalence with interpolation.

Theorem

For fixed distinct nodes $\{x_j\}_{j\in[n]}$, the weights of an interpolatory quadrature rule are identical to those of a moment matching quadrature rule.

Example

The following is a one-point qudarature rule approximation:

$$\int_0^h u(x)\mathrm{d}x \approx h u(0).$$

I.e., $x_1 = 0$ and $w_1 = h$.

The *order of accuracy* of this rule is the constant p in an $\mathcal{O}(h^p)$ estimation of,

$$\int_0^h u(x)\mathrm{d}x - h u(0)$$

What is the order of accuracy of this quadrature rule?

$$u(x) = u(0) + x u'(0) + \frac{u''(0)}{2} x^2 + \cdots$$

$$\int_0^h u(x)\,dx = u(0) h + u'(0) \frac{h^2}{2} + \frac{u''(0)}{2} \cdot \frac{h^3}{6} + \cdots$$

$$\int_0^h u(x)\,dx - h u(0) \sim h^2$$

Example

The following is a one-point qudarature rule approximation:

$$\int_0^h u(x)\mathrm{d}x \approx h u(0).$$

I.e., $x_1 = 0$ and $w_1 = h$.

The *order of accuracy* of this rule is the constant p in an $\mathcal{O}(h^p)$ estimation of,

$$\int_0^h u(x)\mathrm{d}x - h u(0)$$

What is the order of accuracy of this quadrature rule?

An essentially identical quadrature rule approximation is:

$$\frac{1}{h}\int_0^h u(x)\mathrm{d}x \approx u(0).$$

$\frac{1}{h} u(x) \sim \frac{1}{h} u(0) + u'(0) x + \cdots$

$\frac{1}{h}\int_0^h u(x)dx - u(0) \sim \frac{h}{2}$

What is the order of accuracy of this quadrature rule?

Example

Identify the order of accuracy of the quadrature rules,

$$\int_0^h u(x)\mathrm{d}x \approx h u(h). \quad \frac{1}{h}\int_0^h u(x)\mathrm{d}x \approx u(h).$$

$\frac{1}{h}\int_0^h u(x)dx \approx u(h)$

~~$p=3$~~ $p=2$

~~$p=2$~~ $p=1$

Example

Identify the weights and order of accuracy of the quadrature rules,

$$\int_0^h u(x)\mathrm{d}x \approx w_0 u(0) + w_1 u(h)$$

$$\frac{1}{h}\int_0^h u(x)\mathrm{d}x \approx \frac{w_0}{h} u(0) + \frac{w_1}{h} u(h)$$

$$\int_0^h u(x)dx \simeq w_0 u(0) + w_1 u(h)$$

$$\int_0^h x^j dx = w_0 (x^j)\big|_{x=0} + w_1 (x^j)\big|_{x=h} \qquad j=0,1$$

$j=0$: $h = w_0 + w_1$

$j=1$: $\frac{h^2}{2} = w_1 h$

$w_1 = h/2$

$w_0 = h/2$

LeVeque, Randall J. (2007). *Finite Difference Methods for Ordinary and Partial Differential Equations: Steady-State and Time-Dependent Problems*. SIAM. ISBN: 978-0-89871-783-9.