Integration/differentiation with polynomial approximations

MATH 6610 Lecture 27

November 18, 2020

Polynomial approximation

We've seen that polynomial interpolation is unisolvent.

And we've explored the accuracy of polynomial interpolation.

Polynomial approximation

We've seen that polynomial interpolation is unisolvent.

And we've explored the accuracy of polynomial interpolation.

However, our main goal is utilizing polynomial methods for integration and differentiation.

Polynomial approximation

We've seen that polynomial interpolation is unisolvent.

And we've explored the accuracy of polynomial interpolation.

However, our main goal is utilizing polynomial methods for integration and differentiation.

The basic ideas:

- Given function data at some points, we can construct a polynomial interpolant.
- (Interpolatory quadrature) Integration: we approximate the integral of the function as the integral of the polynomial interpolant.
- (Finite differences) Differentiation: we approximate the derivative of the function as the derivative of the polynomial interpolant.

Interpolatory quadrature

With $x_1, \ldots, x_n$ unique, fixed nodes on $[a, b]$, and f a given continuous function: Define $I_{n-1} : C([a, b]) \to P_{n-1}$ as the interpolation operator:

$$I_{n-1} f := \sum_{j=1}^{n} f(x_j) \ell_j(x), \qquad \ell_j(x) := \prod_{\substack{k=1,\ldots,n \\ k \neq j}} \frac{x - x_k}{x_j - x_k}.$$

$$= \sum_{j=1}^{n} c_j x^{j-1}, \qquad Vc = f.$$

Interpolatory quadrature

With $x_1, \ldots, x_n$ unique, fixed nodes on $[a,b]$, and f a given continuous function: Define $I_{n-1} : C([a,b]) \to P_{n-1}$ as the interpolation operator:

$$I_{n-1} f := \sum_{j=1}^{n} f(x_j) \ell_j(x), \qquad \ell_j(x) := \prod_{\substack{k=1,\ldots,n \\ k \neq j}} \frac{x - x_k}{x_j - x_k}.$$

$$= \sum_{j=1}^{n} c_j x^{j-1}, \qquad Vc = f.$$

An *interpolatory quadrature* rule is a set of nodes $\{x_j\}$ and weights $\{w_j\}$ that results from using the integral of $I_{n-1} f$ to approximate the integral of f:

$$\int_a^b f(x) \mathrm{d}x \approx \int_a^b [I_{n-1} f](x) \mathrm{d}x =: \sum_{j=1}^{n} w_j f(x_j).$$

There are two ways to compute such nodes and weights, depending on the algorithmic strategy for I_{n-1}.

Interpolatory quadrature

With $x_1, \ldots, x_n$ unique, fixed nodes on $[a, b]$, and f a given continuous function: Define $I_{n-1} : C([a, b]) \to P_{n-1}$ as the interpolation operator:

$$I_{n-1} f := \sum_{j=1}^{n} f(x_j) \ell_j(x), \qquad \ell_j(x) := \prod_{\substack{k=1,\ldots,n \\ k \neq j}} \frac{x - x_k}{x_j - x_k}.$$

$$= \sum_{j=1}^{n} c_j x^{j-1}, \qquad Vc = f.$$

An *interpolatory quadrature* rule is a set of nodes $\{x_j\}$ and weights $\{w_j\}$ that results from using the integral of $I_{n-1} f$ to approximate the integral of f:

$$\int_a^b f(x) \mathrm{d}x \approx \int_a^b [I_{n-1} f](x) \mathrm{d}x =: \sum_{j=1}^{n} w_j f(x_j).$$

There are two ways to compute such nodes and weights, depending on the algorithmic strategy for I_{n-1}.

Example

Compute weights for the quadrature rule $\sum_{j=-1}^{1} w_j f(j)$ that is accurate to as high a polynomial degree as possible.

Interpolatory quadrature accuracy

The standard way to compute error estimates for polynomial quadrature: Taylor series arguments.

Typically one utilizes an error estimate for interpolation. For n-point interpolation on $[a, b]$, exact on P_{n-1}:

$$f(x) - [I_{n-1} f](x) = \frac{f^{(n)}(\xi)}{n!} \prod_{j=1}^{n} (x - x_j), \qquad \xi = \xi(x) \in [a, b].$$

Interpolatory quadrature accuracy

The standard way to compute error estimates for polynomial quadrature: Taylor series arguments.

Typically one utilizes an error estimate for interpolation. For n-point interpolation on $[a, b]$, exact on P_{n-1}:

$$f(x) - [I_{n-1}f](x) = \frac{f^{(n)}(\xi)}{n!} \prod_{j=1}^{n} (x - x_j), \qquad \xi = \xi(x) \in [a, b].$$

Chaining this together with an integral:

$$\left| \int_a^b f(x) \mathrm{d}x - \int_a^b [I_{n-1}f](x) \mathrm{d}x \right| \leqslant \frac{(b-a)^{n+1}}{n!} \max_{x \in [a,b]} \left| f^{(n)}(x) \right|.$$

Types of quadrature rules

$$\int_a^b f(x)\mathrm{d}x \approx \sum_{j=1}^{n} w_j f(x_j).$$

There are several special types of quadrature rules:

- Equidistant nodes: "Newton-Cotes" rules
  - "Closed": the endpoints are nodes (e.g., a, b)
  - "Open": all nodes are interior to the interval
- Change-of-variable tricks, e.g., used for infinite-interval quadrature

Types of quadrature rules

$$\int_a^b f(x)\mathrm{d}x \approx \sum_{j=1}^{n} w_j f(x_j).$$

There are several special types of quadrature rules:

- Equidistant nodes: "Newton-Cotes" rules
  - "Closed": the endpoints are nodes (e.g., a, b)
  - "Open": all nodes are interior to the interval
- Change-of-variable tricks, e.g., used for infinite-interval quadrature
- Chebyshev/arcsine-like nodal arrangements: Clenshaw-Curtis quadrature, Fejér quadrature
- Hermite quadrature: uses derivatives as well as function values

Types of quadrature rules

$$\int_a^b f(x)\mathrm{d}x \approx \sum_{j=1}^{n} w_j f(x_j).$$

There are several special types of quadrature rules:

- Equidistant nodes: "Newton-Cotes" rules
  - "Closed": the endpoints are nodes (e.g., a, b)
  - "Open": all nodes are interior to the interval
- Change-of-variable tricks, e.g., used for infinite-interval quadrature
- Chebyshev/arcsine-like nodal arrangements: Clenshaw-Curtis quadrature, Fejér quadrature
- Hermite quadrature: uses derivatives as well as function values
- Gaussian quadrature: Fixing n, has the maximum possible polynomial degree of exactness.

Finite difference formulas

With $x_1, \ldots, x_n$ unique, fixed nodes on $[a,b]$, and f a given continuous function: Define $I_{n-1} : C([a,b]) \to P_{n-1}$ as the interpolation operator:

$$I_{n-1} f := \sum_{j=1}^{n} f(x_j) \ell_j(x), \qquad \ell_j(x) := \prod_{\substack{k=1,\ldots,n \\ k \neq j}} \frac{x - x_k}{x_j - x_k}.$$

$$= \sum_{j=1}^{n} c_j x^{j-1}, \qquad Vc = f.$$

Finite difference formulas

With $x_1, \ldots, x_n$ unique, fixed nodes on $[a,b]$, and f a given continuous function: Define $I_{n-1} : C([a,b]) \to P_{n-1}$ as the interpolation operator:

$$I_{n-1} f := \sum_{j=1}^{n} f(x_j)\ell_j(x), \qquad \ell_j(x) := \prod_{\substack{k=1,\ldots,n \\ k\neq j}} \frac{x - x_k}{x_j - x_k}.$$

$$= \sum_{j=1}^{n} c_j x^{j-1}, \qquad Vc = f.$$

A *finite difference formula* rule is a set of nodes $\{x_j\}$ and weights $\{w_j\}$ that results from using the derivative of $[I_{n-1} f](x)$ to approximate $f'(x)$.
Some minor differences from the quadrature case:

- The weights depend on the location x.
- The weights can be defined by means other than interpolation.

Again, there are multiple ways to compute weights depending on how the interpolant is identified.

Finite difference formulas

With $x_1, \ldots, x_n$ unique, fixed nodes on $[a, b]$, and f a given continuous function:
Define $I_{n-1} : C([a, b]) \to P_{n-1}$ as the interpolation operator:

$$I_{n-1} f := \sum_{j=1}^{n} f(x_j) \ell_j(x), \qquad \ell_j(x) := \prod_{\substack{k=1,\ldots,n \\ k \neq j}} \frac{x - x_k}{x_j - x_k}.$$

$$= \sum_{j=1}^{n} c_j x^{j-1}, \qquad Vc = f.$$

A *finite difference formula* rule is a set of nodes $\{x_j\}$ and weights $\{w_j\}$ that results from using the derivative of $[I_{n-1} f](x)$ to approximate $f'(x)$.
Some minor differences from the quadrature case:

- The weights depend on the location x.
- The weights can be defined by means other than interpolation.

Again, there are multiple ways to compute weights depending on how the interpolant is identified.

Example

Compute a 3-point finite-difference formula on the nodes $\{-1, 0, 1\}$ for differentiation at $x = 0$.

Order of accuracy

Error estimates for finite difference formulas are computable via Taylor's Theorem. Recall that interpolation error on the interval $[a, b]$ scales like $(b-a)^{n+1}$.

Thus, finite difference formulas become more accurate as $(b-a) \to 0$.

Order of accuracy

Error estimates for finite difference formulas are computable via Taylor's Theorem. Recall that interpolation error on the interval $[a, b]$ scales like $(b-a)^{n+1}$.

Thus, finite difference formulas become more accurate as $(b-a) \to 0$.

To standardize this, we typically assume *equidistant* nodes, with a spacing of $h > 0$. The *order of accuracy* of a finite difference formula is typically $\mathcal{O}(h^k)$ for some integer k.

Order of accuracy

Error estimates for finite difference formulas are computable via Taylor's Theorem. Recall that interpolation error on the interval $[a, b]$ scales like $(b - a)^{n+1}$.

Thus, finite difference formulas become more accurate as $(b - a) \to 0$.

To standardize this, we typically assume *equidistant* nodes, with a spacing of $h > 0$. The *order of accuracy* of a finite difference formula is typically $\mathcal{O}(h^k)$ for some integer k.

Example

Compute a finite difference formula and order of accuracy (error estimate) for the three finite difference formulas:

$$f'(x) \approx w_1 f(x) + w_2 f(x + h)$$
$$f'(x) \approx w_1 f(x) + w_2 f(x - h)$$
$$f'(x) \approx w_1 f(x + h) + w_2 f(x - h)$$

Order of accuracy

Error estimates for finite difference formulas are computable via Taylor's Theorem. Recall that interpolation error on the interval $[a, b]$ scales like $(b-a)^{n+1}$.

Thus, finite difference formulas become more accurate as $(b-a) \to 0$.

To standardize this, we typically assume *equidistant* nodes, with a spacing of $h > 0$. The *order of accuracy* of a finite difference formula is typically $\mathcal{O}(h^k)$ for some integer k.

Example

Compute a finite difference formula and order of accuracy (error estimate) for the three finite difference formulas:

$$
\begin{aligned}
f'(x) &\approx w_1 f(x) + w_2 f(x+h) \\
f'(x) &\approx w_1 f(x) + w_2 f(x-h) \\
f'(x) &\approx w_1 f(x+h) + w_2 f(x-h)
\end{aligned}
$$

Odds and ends: higher-order derivatives can be approximated, function derivatives instead of function values can be used, etc.