Review Sheet for Midterm 2 Selected Solutions APPM/MATH 4650 Fall '20 Numerical Analysis

Instructor: Prof. Becker
solutions version 10/26/2020

Chapter 3: interpolation, mainly Hermite interpolation and cubic spline interpolation

1. Can be bound the error of approximating a function f by its unique degree n-or-less interpolant p (on a specified set of $n+1$ nodes)?

   Solution:

   Yes, there is, as long as f is smooth enough. Here's one:

   Theorem 3.3 Suppose $x_0, x_1, \ldots, x_n$ are distinct numbers in the interval $[a, b]$ and $f \in C^{n+1}[a, b]$. Then, for each x in $[a, b]$, a number $\xi(x)$ (generally unknown) between $\min\{x_0, x_1, \ldots, x_n\}$, and the $\max\{x_0, x_1, \ldots, x_n\}$and hence in (a, b), exists with

   $$f(x) = P(x) + \frac{f^{(n+1)}(\xi(x))}{(n+1)!}(x - x_0)(x - x_1)\cdots(x - x_n), \tag{3.3}$$

   where $P(x)$ is the interpolating polynomial given in Eq. (3.1).

   There are other ways that the error term for the Lagrange polynomial can be expressed, but this is the most useful form and the one that most closely agrees with the standard Taylor polynomial error form.

   Proof Note first that if $x = x_k$, for any $k = 0, 1, \ldots, n$, then $f(x_k) = P(x_k)$, and choosing $\xi(x_k)$ arbitrarily in (a, b) yields Eq. (3.3).

2. What's the idea of Hermite interpolation? Can this be done in a piecewise (composite) fashion?

   Solution:

   It's like usual interpolation that fits an interpolant p to match $f(x_i)$ at a set of nodes $\{x_i\}$, but it also matches $p'(x_i) = f'(x_i)$. It can be done in a piecewise fashion (in fact, it usually is)

3. Briefly, what's the difference between Hermite interpolation and cubic splines?

   Solution:

   Hermite interpolation matches $p'(x_i) = f'(x_i)$, whereas cubic splines enforce consistency so that the interpolant p, which is piecewise cubic (and hence not necessarily smooth) has at least 2 continuous derivatives, i.e., making sure the derivatives and second derivatives match each other at the nodes. It does not require them to match the derivatives of f.

Chapter 4, part 1: numerical differentiation, Richardson extrapolation

1. How are finite difference formulas usually derived?

   Solution:

   By using the given nodes (specified by the desired type of formula, like centered-difference or forward-difference) to interpolate with a polynomial, and then differentiating that polynomial.

2. What is the basic forward difference formula? What's the order of error? What is the basic centered difference formula? What's the order of error?

Solution:

For forward differences, it is $\frac{f(x+h)-f(x)}{h}$, and it's $O(h)$ error.

For centered differences, it is $\frac{f(x+h)-f(x-h)}{2h}$, and it's $O(h^2)$ error.

3. Why might we prefer a higher-order method?

Solution:

We can pick a larger h and not have as much roundoff error

4. Why don't we use order 100 methods?

Solution:

There's little advantage after going to about 6th order — the gains in overall accuracy are slight, since, as a rule-of-thumb, we're always bounded by machine epsilon for relative error. It requires more function evaluations to run higher-order methods.

5. What's the idea behind Richardson extrapolation?

Solution:

If we have an approximation that is $O(h^p)$, and we know p, then we can evaluate this approximate scheme at two values of h, usually h and $2h$, and then combine these approximations to cancel off the $O(h^p)$ term.

6. Let $F : \mathbb{R}^m \to \mathbb{R}^n$ be a vector-valued function, with domain $\mathbb{R}^m$ and range $\mathbb{R}^n$. For which values of m and n is automatic differentiation (AD) really useful? For which values are finite difference rules a good idea? What are the downsides of AD?

Solution:

If m is very large, $m \gg n$, then *reverse mode AD* is the best option, and takes $O(n)$ function evaluations. This is what backpropagation for training neural nets does.

If n is very large, $n \gg m$, then *forward mode AD* is the best option, and takes $O(m)$ function evaluations.

If $n = m = 1$ (or both dimensions are small), the finite difference methods work well (as do AD). Finite difference methods are generally easier to work with, as AD requires specialized libraries/environments (e.g., tensorflow, pytorch, Dolfin adjoint, JAX for Python; Zygote.jl for Julia; AdiGator for Matlab). Even in 1D, it's possible for reverse-mode AD has a memory explosion, while that's not possible for finite difference methods.

7. For cubic splines on $n + 1$ nodes, how many parameters are there?

Solution:

We have n regions, and each region has 4 parameters since we represent a region with a cubic polynomial, so $4n$ overall parameters.

8. For cubic splines on $n + 1$ nodes, how many conditions to satisfy are there?

Solution:

How many conditions do we have? Here's one way to count: We have $n - 1$ interior nodes, and the function values, the first, and the second derivatives must match here, so that's $3(n - 1)$ conditions. We have $n + 1$ nodes overall, and need to interpolate the original function, so that's another $n + 1$ conditions, for a total of $4n - 2$. We add 2 more conditions as boundary conditions (e.g., "natural" conditions, "clamped" conditions, "not-a-knot" conditions, "periodic" conditions, etc.)

9. Running not-a-knot cubic spline interpolation on $f(x) = |x|$ on $[-1, 1]$, in which region of the domain do we expect the error to be the highest?

Solution:

At $x = 0$, since f is the least smooth there

10. Running natural cubic spline interpolation on $f(x) = x^2$ on $[-1, 1]$, in which region of the domain do we expect the error to be the highest?

Solution:

At the boundaries near -1 and 1: this is the Runge phenomenon.

Chapter 4, part 2: numerical integration

1. How are quadrature formulas usually derived?

Solution:

By using the nodes to interpolate with a polynomial, and then differentiating that polynomial.

2. For quadrature, do we have a notion of "forward" vs "centered" formulas?

Solution:

No. Differentiation is done at a point, say x, so we have a notion of adding nodes forward, behind or centered around x. For integration, we never integrate "at" a point x, we always integrate "over" an interval $[a, b]$.

3. What does Newton-Cotes mean? Are all quadrature rules Newton-Cotes?

Solution:

Newton-Cotes formulas mean (1) we use equispaced nodes, and (2) we interpolate with a polynomial of the appropriate degree (i.e., with $n + 1$ nodes, we use the degree n interpolating polynomial). Not all quadrature rules are Newton-Cotes. Most quadrature rules follow (2), or follow (2) in a piecewise sense (so "composite" rules); we call these rules "interpolatory". Some rules, like Gaussian quadrature rules, do not use equispaced nodes; also, adaptive quadrature rules use non-equispaced rules.

4. Is it OK to use a higher-order Newton-Cotes formula rather than switching over to a composite rule?

Solution:

No! The formulas become annoying to derive as $n \to \infty$, and they are numerically unstable as we take the limit of increasing n (due to the Runge phenomenon). Much better to switch to composite rules if you need increased accuracy (well, in fact, almost no one ever uses a plan Newton-Cotes, we almost always use a composite Newton-Cotes)

5. What's the idea behind Gauss-Legendre integration?

Solution:

By choosing unequispaced nodes, chosen as the roots of the n^{th} Legendre polynomial, we get much higher accuracy, as quantified by having a higher order of exactness.

6. What does it means to say the "order of exactness/precision" of a quadrature rule?

Solution:

If the order of exactness is k, it means the quadrature rule *exactly* integrates all polynomials of degree k or less.

7. What's the order of exactness of Newton-Cotes methods?

Solution:

It depends on whether n is even or odd, where we use $n + 1$ nodes (since we like to list them as $x_0, \ldots, x_n$). For odd n, it's order n. For even n, it's order $n + 1$.

8. What's the order of exactness of the Gauss-Legendre method?

Solution:

For Gauss-Legendre, we usually work with n nodes (labeling them $x_1, \ldots, x_n$), and we have order of exactness $2n - 1$.

9. What's the difference between open and closed Newton-Cotes formulas? Give an example of each

Solution:

Open means the nodes do not include the end-point, closed means we do. For open nodes, $h = \frac{b-a}{n+2}$, whereas $h = \frac{b-a}{n}$ for closed nodes. The midpoint rule is an example of an open formula; the trapezoidal and Simpson rules are examples of closed formulas.

10. Finite difference formulas are unstable; we cannot take $h \to 0$ and get error $\to 0$. What about composite Newton-Cotes? If we take $n \to \infty$, does error go to zero?

Solution:

Composite Newton-Cotes methods are stable; the error does not go to zero (it goes to machine epsilon), but that's pretty good. Unlike finite difference formulas, the error does not get worse as we increase n (or decrease h).

11. What are the relevant properties of the Legendre polynomials? Are these the same as Lagrange polynomials?

Solution:

The n^{th} Legendre polynomial L_n is a degree n monic polynomial, and it is orthogonal to L_m for all $m \neq n$, meaning

$$\int_{-1}^{1} L_n(x) L_m(x)\, dx = 0$$

And no, these are not the same as Lagrange polynomials. Those are used for interpolation!

12. Gauss-Legendre integration only works on $[-1, 1]$

Solution:

I wouldn't ask a question this vague on the exam; the answer is True or False depending on how you interpret it. Basic Gauss-Legendre does only work on $[-1, 1]$ (so True), but if we're on $[a, b]$, it's easy to do a change of variables to work in $[-1, 1]$ (so False).

13. For each L-named French mathematician in the following list, list the corresponding topic named after them that we've seen in our course: Legendre, Lagrange, Laguerre, L'Hôpital. Bonus: Laplace (not yet covered in this class), and Lipschitz (actually a German)

Solution:

a) Legendre: for Legendre polynomials and Gauss-Legendre integration; this is a family of orthogonal polynomials on $[-1, 1]$ with respect to the weight $w(x) = 1$.

b) Lagrange: for constructing the interpolating polynomial

c) Laguerre: for Laguerre polynomials and Gauss-Laguerre integration; this is a family of orthogonal polynomials on $[0, \infty]$ with respect to the weight $w(x) = e^{-x}$.

d) L'Hôpital: for L'Hôpital's rule for taking limits

e) Laplace: many things (none yet covered in this class), such as the Laplace equation and the Laplacian

f) Lipschitz: for Lipschitz continuity

14. What are the relevant properties of the Laguerre polynomials?

Solution:

The n^{th} Laguerre polynomial G_n is a degree n monic polynomial, and it is orthogonal to G_m for all $m \neq n$ in a *weighted sense*, meaning

$$\int_0^\infty G_n(x)G_m(x)e^{-x}\,dx = 0$$

15. How are Hermite interpolation and Gauss-Hermite quadrature related?

Solution:

They aren't! Same mathematician Hermite, but completely different concepts.

16. The same Romberg integration formula works for both composite midpoint and composite trapezoid rules

Solution:

No. Both midpoint and trapezoid are $O(h^2)$, but midpoint is $O(h^2)+O(h^3)+O(h^4)+O(h^5)\ldots$ whereas trapezoid is $O(h^2) + O(h^4) + O(h^6) + O(h^8)\ldots$ (via Euler-Maclaurin), so the Romberg integration formulas are different since they must take this into account,

17. Is there any limit to how many times we can apply Romberg integration?

Solution:

Well, to apply it k times, using the standard factor of $h/2$ to go from one row to the next, requires we have at least 2^k nodes, so that's a lot of nodes if k is large. Furthermore, if the integrand isn't sufficiently smooth, at some point the error formula will break down. Finally, we often get to machine precision quickly, so then there's no point in going further.

18. For a 2D integral, we need to rederive quadrature rules

Solution:

False, we can re-use our 1D quadrature rules

19. If a quadrature rule in d dimensions, then in terms of the spacing h between node points, how expensive to we expect the rule to be? like d^h, h^{-d}, e^d/h, hd, etc?

Solution:

We expect h^{-d}. This is very expensive for large dimensions!

20. Monte Carlo integration is faster than quadrature rules

Solution:

Not always true: it's much slower in low dimensions (since for Monte Carlo, it takes a VERY long time to reach accuracies on the order of 10^{-16}), but it's not affected by dimension, so in large dimensions, Monte Carlo is the only viable option since quadrature rules become infeasible.

21. Basic composite Newton-Cotes formulas cover proper integrals like $\int_0^1 1/x\,dx$ (true/false?)

Solution:

False: first, this is not a proper integral since the integrand is unbounded. Secondly, composite Newton-Cotes will not work, we need to do one of the tricks we've talked about.

22. Gauss-Laguerre and Gauss-Hermite formulas can be used to accurately integrand any function over $[0, \infty)$ or $(-\infty, \infty)$, respectively.

Solution:

False, they can be used in theory, but we may run into serious numerical issues, as we saw on the homework. To integrate $\int_0^\infty f(x)\,dx$ via Gauss-Laguerre, for example, we have to write this as $\int_0^\infty \varphi(x)e^{-x}\,dx$ for $\varphi(x) = e^x f(x)$. If $f(x)$ doesn't decay exponentially fast already, then $\varphi(x)$ may grow extremely large and lead to overflow. Even if it doesn't grow large, we need it to decay quickly so that we don't need too many Gauss-Laguerre weights, since the actual weights for Gauss-Laguerre have underflow if you need to go out too far.

23. The Runge phenomenon affects... interpolation? differentiation? integration? When does it arise?

Solution:

It affects all of them, though we don't usually see it in differentiation since we don't do high-order equispaced finite difference rules (either we use *composite* rules, or Gassian quadrature, which is not equispaced).

It affects both interpolation and integration; because it affects interpolation, and since our integration rules are based off of integrating the interpolant, therefore it affects integration too.

It arises when we have equispaced nodes, and manifests itself as having large error near the boundaries. It goes away if we correctly add enough nodes near the boundaries, which is what Gauss-Legendre integration does.

Chapter 5: IVPs and ODEs

1. We talk about IVP for first-order ODE. Why don't we talk about boundary value problems (BVP) for first-order ODE?

Solution:

BVP have two boundary conditions, but a first-order ODE only takes one condition, so we can't have a BVP (if we specify the end point as the boundary condition, then we can flip the sign of t and treat it as an initial condition). We only talk about BVP for second-order and higher ODE.

2. The wave equation is an example of an ODE

Solution:

False, it's a partial differential equation (PDE), and not covered in this chapter.

3. All ODEs have analytic solutions, unlike for PDE

Solution:

This is not true in the sense that it is not true that all ODEs have analytic solutions. However, it is true that neither do most PDE have analytic solutions.

4. When running a high-order ODE solver, we get a list of independent variable points $\{t_i\}$ and corresponding points $\{w_i\}$ that approximate $w_i \approx y(t_i)$, where y is the true solution to the IVP. When plotting the solution, if `t` is the vector of $\{t_i\}$ and `w` is the vector of $\{w_i\}$, we get a good idea of how y behaves by plotting `plot(t,w,'-')` (either via Matlab or via Python with `from matplotlib.pyplot import plot`) (true/false?)

Solution:

False. For using Euler's method, this captures y up to the order on which we've solved the ODE, but for higher-order methods, this style plot is a piecewise linear interpolation, and misses the intermediate behavior of y which can be accurately approximated via piecewise Hermite interpolation.

5. Just like for multidimensional integration, numerically solving a coupled system of d ODEs is way harder than solving a single system.

Solution:

Not true; we do have to deal with potentially stiff problems, but otherwise solving a system of equations is straightforward and not much more expensive (about $d\times$ more expensive than a single IVP, rather than to the d^{th} power as we have in integration).

6. What is the local truncation error of Euler's method (aka Forward Euler)? What is the global error?

Solution:

Both are $O(h)$.